A REVIEW OF MEMORY ALLOCATION AND MANAGEMENT IN COMPUTER SYSTEMS

Ahmed Faraz

Department of Computer & Software Engineering, Bahria University Karachi Campus, 13 National Stadium Road, Karachi -75260, Pakistan

ABSTRACT

In this paper I have described the memory management and allocation techniques in computer systems. The purpose of writing this survey paper is to elaborate the concept of memory allocation and management in computer systems because of the significance of memory component in computer system's hardware. It is apparent from the work of computer scientists that effective and efficient main memory management and virtual memory management in computer systems improves the computer system's performance by increasing throughput and processor utilization and by decreasing the response time and turnaround time. Firstly I have compared Uniprogramming system with Multiprogramming system. After comparison I found that Multiprogramming systems are quite better than Uniprogramming systems from the point of view of memory utilization. Also the functionality of operating system routines which are responsible for user's memory partitioning must be improved to get better system performance in Multiprogramming system .In Uniprogramming system , the processor remains idle most of the time but dividing the memory into partitions for holding multiple processes as in Multiprogramming system does not solve the problem of idleness of a processor. Mostly all of the processes need I/O access, therefore processor also remain idle in Multiprogramming system. We have also discussed resource memory in detail and compared fixed partitioning with variable partitioning. After in depth study we found that variable partitioning is more advantageous than fixed partitioning because reallocation of page frames is impossible in fixed partitioning for a set of active processes at time instant't'. In this paper we have also discussed MIPS R2/3000 machine virtual to real address mapping in detail so that virtual to real address mapping can be understood through a machine's architecture and example.

KEYWORDS

Queues, Long Term Queues, Short Term Queues, I/O Queues, Swapping, Processor Scheduling, Fixed Partitioning, Variable Partitioning, Page Frames, Pages, Virtual Memory, Physical Memory, Processor Utilization, Throughput, Response Time, Turnaround Time

1. INTRODUCTION

There are different models and classification of memory management schemes. The computer systems deal with allocation of memory pages to processes. In order to allocate the memory pages to active processes, two policies or schemes are implemented in computer systems. These policies or schemes are "Fixed Partitioning" and "Variable Partitioning". These policies are also referred as Fixed Allocation Scheme and Variable Allocation Scheme. In Fixed Partitioning, the partition of memory apportioned to an active process is unalterable during the period of existence of a process. In Variable Partitioning, the partition of memory apportioned to an active process is alterable according to the memory requirements of an active process during the period of existence of a process. In this paper we reviewed memory management in computer systems and discussed a case study of MIPS R2/3000 virtual to real address mapping at the end of the paper so that an advanced computer system's memory management can serve as an example for understanding computer system's memory management.

2. MEMORY MANAGEMENT

Usually there are two types of systems, Uniprogramming systems and Multiprogramming systems. The Uniprogramming systems are those systems in which memory is divided into two parts. The first part of the memory contains operating system and the second part of memory contains the programs which are currently being executed. The first part of the memory which contains operating system is called "Resident Memory". The Multiprogramming systems are those systems in which memory is divided into two parts .The first part of the memory contains operating system and the second part of the memory contains the programs which are currently being executed as discussed in [23]. Here in Multiprogramming systems, the user part of the memory is further subdivided into multiple sub-parts or sub-portions. These sub-parts or sub-portions are used to accommodate multiple processes. It is the operating system that subdivides user's memory into sub-parts or sub-portions to accommodate multiple processes.

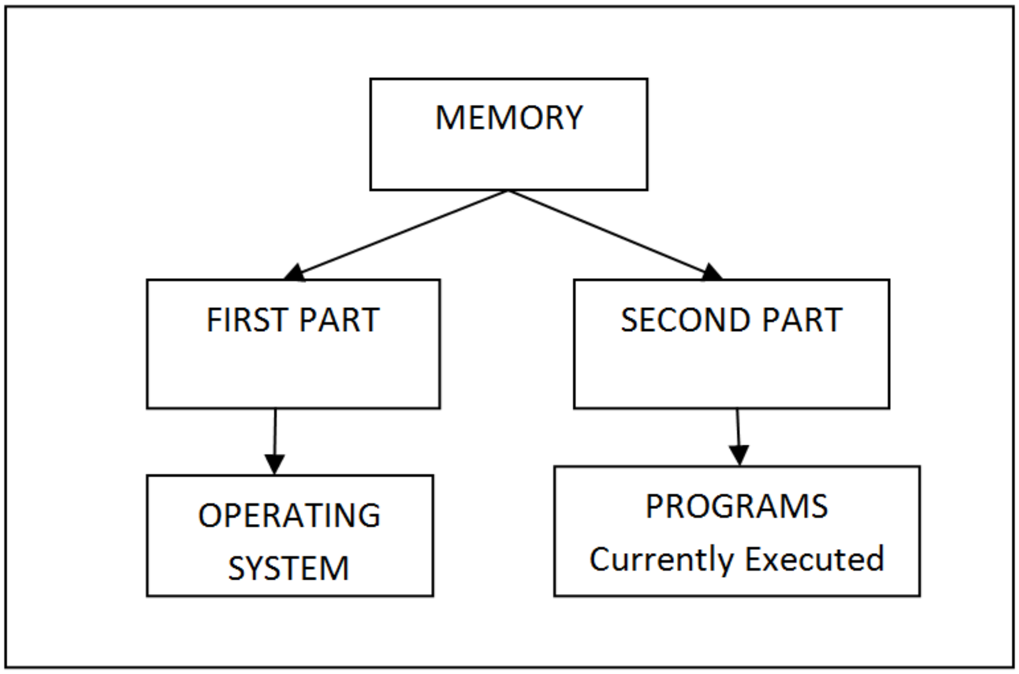

Figure 1. Memory Management in Uniprogramming System

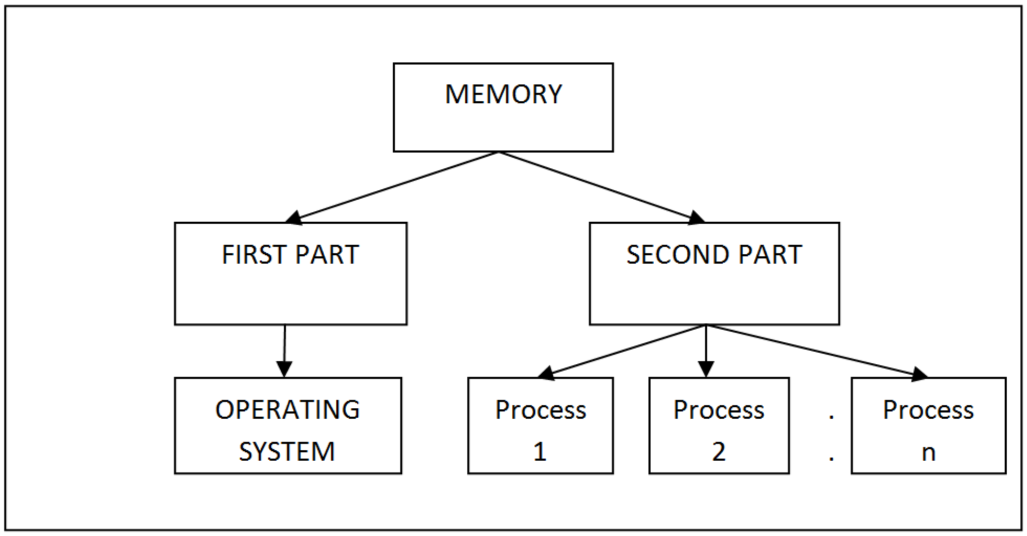

Figure 2. Memory Management in Multiprogramming System

2.1. Queues in Computer Systems

Queues are defined as waiting lines and are implemented as buffers in computer systems. These buffers are either on primary storage medium that is main memory or they are on secondary storage medium that is hard disk. When processes are ready to execute, they are loaded in memory and are allocated a dedicated memory space for execution. On the basis of readiness for execution, different queues are implemented in memory of computer system and on the secondary storage medium or hard disk as elaborated in [23].

Those processes which reside in I/O queues have to wait for a long time because the execution of I/O processes in memory of computer system depend upon the readiness of I/O device .We have observed that the processor in uniprogramming system remains idle most of the time because I/O activities are much slower than computation. In Uniprogramming system only one process is executed in memory at one time, therefore if the process is waiting for I/O access then the processor cannot perform any action or execution and has to wait. Due to this waiting for I/O device access, the processor goes into dormant state; therefore processor utilization in Uniprogramming system is very low whereas the processor utilization in multiprogramming system is comparatively high because in Multiprogramming system the processor can perform another computation if one process or all of the processes are waiting for I/O device access as discussed in [23].

2.2. Types of Queues

There are three types of queues in computer systems:
(i) Long Term Queues
(ii) Short Term Queues
(iii) I/O Queues

The Long Term Queues contain new processes which arrive in the system for execution. The Long Term Queues are implemented on hard disk space or the secondary storage medium. The Short Term Queues contain those processes which are ready for execution and ready to use the processor. The I/O Queues contain those processes which are not ready to use the processor and these processes require I/O device access for their complete execution.

Defining Long Term Queues for new processes, Short Term Queues for ready processes, I/O Queues for those processes not ready for use by the processor does not solve the problem in entirety. In Multiprogramming system, memory contains multiple processes and processor can move from one process to another process when one process is waiting for I/O. In Multiprogramming system it is very common that all of the processes wait for I/O access and the memory does not hold such processes which are entirely independent of I/O. Therefore, the processor has to wait. The other reason for idleness of the processor in Multiprogramming system is that there is a speed mismatch between processor and I/O devices, the processors are much faster than I/O devices, therefore the processor has to wait after execution of one process. Hence it can be concluded that the processor still remain idle in Multiprogramming system most of the time.

If the size of memory is increased, then it can occupy more processes and more processes can run in memory in Multiprogramming system as discussed in [1]. Nowadays the size of memory is not an issue, random access memory in gigabytes are available at low cost but as the big sized memories are easily available at very low cost, the size of programs and processes also increased too much. When processes of very large size are loaded in memory, the processor takes time in execution of such big processes. For example, those processes or programs written in Java language occupy huge amount of memory and the system becomes slower when Java based applications run.

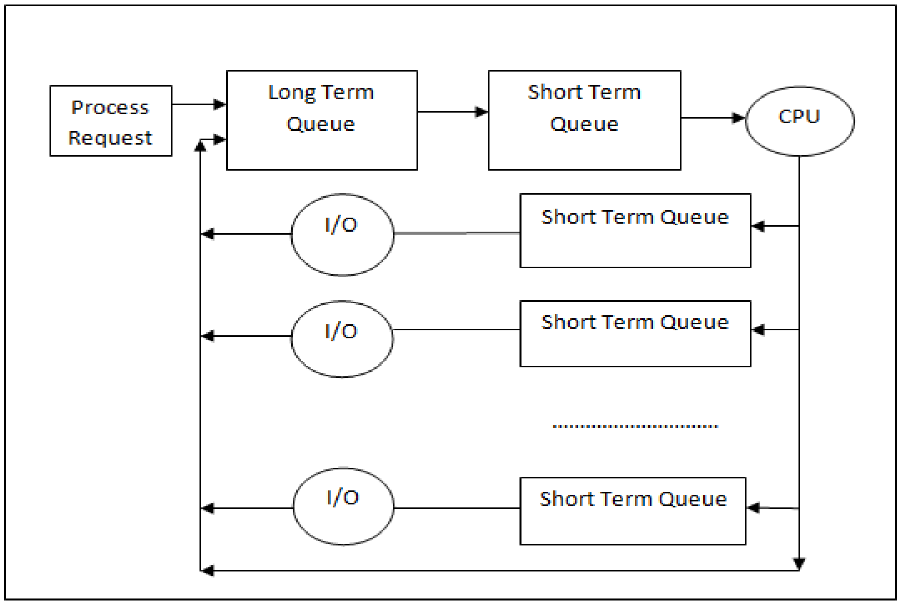

Figure 3. Implementation of Queues in the Processor Scheduling

2.3. Swapping

Since we know that the long term queue is established on disk storage or secondary storage medium and it contains process requests arrived from the system for execution. The process requests are entered into main memory one at a time. Only one process can be executed in main memory at one time instant. When the process completes its execution in main memory, it leaves the main memory. Usually there are two types of processes in main memory: the first type of processes are those processes which complete execution in main memory and the second type of processes are those processes which does not complete execution in main memory [4]. The second type of processes which does not complete their execution are halted but does not enter into the idle state in swapping because the processor establishes named as an “Intermediate Queue” on the disk storage. Those processes which are halted after some partly executions in main memory are brought into the Intermediate Queue from main memory. Now the processor brings a new process into main memory either from an Intermediate Queue or from Long Term Queue for execution. Now the execution continues with the newly arrived process as explained in [23].

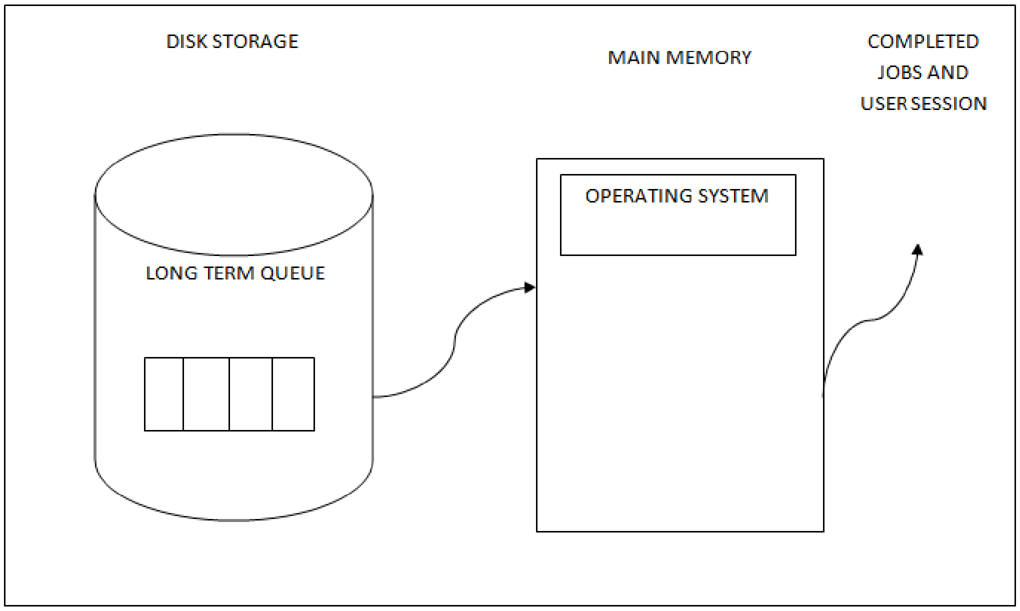

Figure 4. Simple Job Scheduling

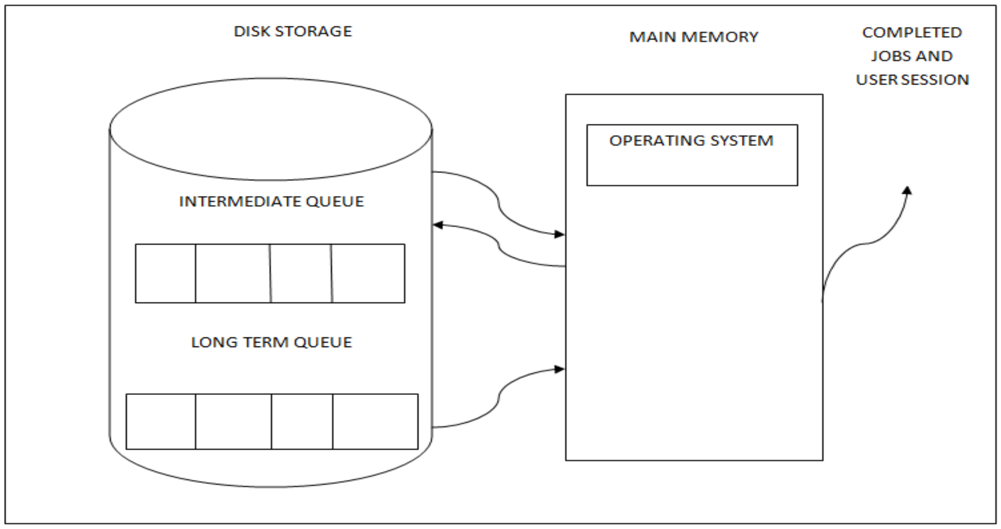

Figure 5. Swapping

2.3.1. Swapping is an I/O Operation, how?

There are two components of Computer Systems which involve in I/O, the disk storage and the main memory. In swapping, the processor brings a process from the long term queue into main memory which is an input operation. This process completes its execution in main memory and leaves the main memory. Those processes which does not complete their execution in main memory are brought from the main memory into intermediate queue which is an output operation and then the processor either selects a process from an Intermediate Queue or from Long Term Queue and bring the selected process into main memory for execution which is an input operation.

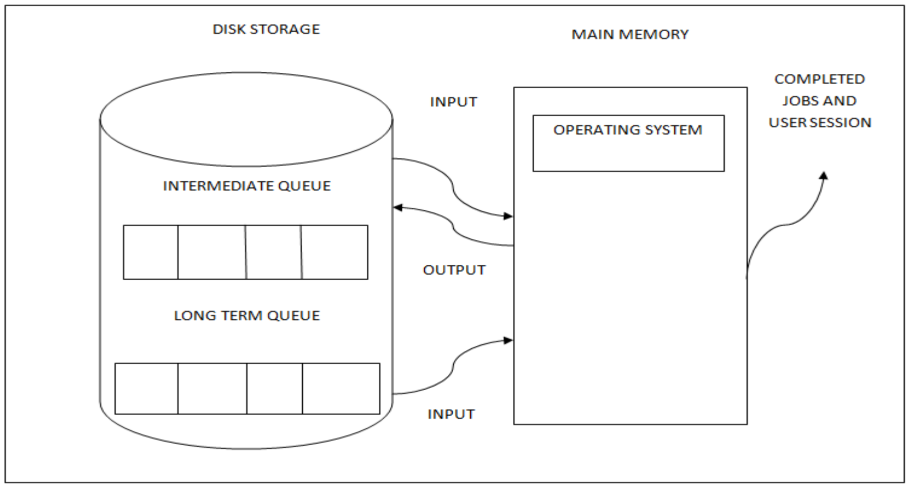

Figure 6. Swapping with Input and Output Operations

2.3.2. Swapping enhances the Computer System's Performance, how?

When we compare disk I/O with tape I/O or printer I/O, we learn that the disk I/O is the fastest I/O on the system .In swapping, disk I/O is involved which is the fastest I/O on the system, therefore swapping enhances the Computer System's performance. The virtual memory improves the Computer System's performance over simple swapping as discussed in [16].

2.4. Classification of Memory Policies

When we discuss Memory Policies Classification, a question is raised in our minds that "what is page fault rate?" In general the page fault rate "f" is a value not entirely intrinsic to the process." Page fault rate "f" is not a parameter pertaining to the real nature or essence of a process. This discriminating and crucial parameter is dependent on memory management policy. Page fault rate "f" defines two things: (a) how many main memory pages are apportioned to a process? (b) Selection of a policy to decide which process's page will reside in main memory as discussed in [16].

2.5. The Effect of Memory Management Policy

Suppose that we have a typical computer system in which there is a change in memory management policy. This change can improve the page-fault rate "f" without changing the system load and other system parameters as discussed in [22]. The following three advantages can be obtained through change in memory management policy:

(i) Improvement in processor utilization
(ii) Increment in system throughput
(iii) Decrement in response time

If someone is so inquisitive and has a question in mind that whether the change in memory management policy may improve the processing efficiency or not, then the reply is that this change does not increase process-paging rate as discussed in [18]. This means that either the process-paging rate remains same or the process-paging rate is decreased due to the change in memory management policy. There are two classes of memory management policies in multiprogramming systems, the first one is Fixed Partitioning and the second one is Variable Partitioning.

2.6. Mathematical Representation

In this survey, the terminologies and mathematical notations are taken from original work [22] by Denning and Graham. Let us consider the interval in which the level of multiprogramming is fixed$d = [d(t)]$. During this interval, the set of active processes will be following:

$$A = \{P_1, P_2, P_3, \dots\dots\dots\dots, P_d\} \quad \text{.................} (1)$$

At time instant't', the process P_i is associated with its resident set$Z_i(t)$. $Z_i(t)$ is defined as the set of the page frames of the process present in memory. The resident set $Z_i(t)$ contains $z_i(t) \geq 1$ pages. This means that the minimum numbers of pages present in the set of the page frames of the process present in main memory must be equal to or greater than 1.There exists sharing among ‘active processes’ in the memory. Due to this reason resident sets $Z_i(t)s$ overlap with each other. The reasoning is that when the processes become active in the memory, then these active processes share data and information with each other. A specific process P_1does not stay in memory without communication with the other processP_2. That is the communication and co-ordination between two processes $P_1$and $P_2$is indispensable. Since the resident sets $Z_i(t)s$are associated with active processes$P_i\ s$, therefore there exists an overlapping between resident sets $Z_i(t)s$and active processes $P_i\ s$ as discussed in [22].

.

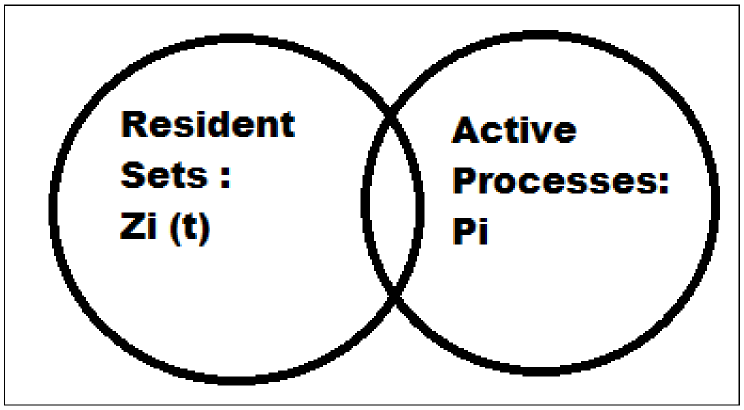

Figure 7. Overlapping of Resident Sets and Active Processes

Finally the resident sets $Z_i(t)$s overlap because of the sharing that takes place among active processes $P_i\ s$

.

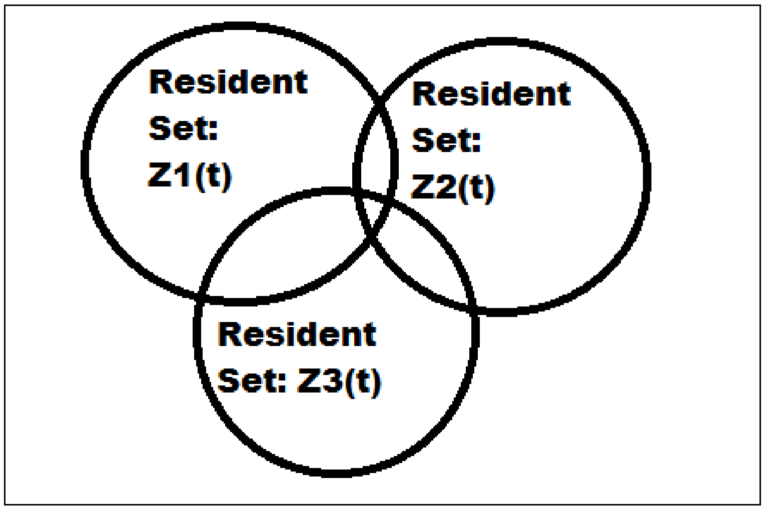

Figure 8. Overlapping of Resident Sets $Z_i(t)$

2.6.1. Partition Vector Z (t)

The management configuration is represented by a Partition Vector Z (t).

$$Z(t) = [Z_1(t), Z_2(t), Z_3(t), \ldots\ldots\ldots\ldots, Z_d(t)] \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (2)$$

This implies that the Partition Vector $Z(t)$consists of a set of d resident sets. Here $Z_1(t)$is the First Resident Set, $Z_2(t)$ is the Second Resident Set, $Z_3(t)$ is the Third Resident Set, and $Z_d(t)$ is the d^{th} Resident Set.

2.6.2. Size Vector z (t)

The Size Vector z (t) is mathematically represented as follows:

$$z(t) = [z_1(t), z_2(t), z_3(t), \ldots\ldots\ldots\ldots, z_d(t)] \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ (3)$$

2.6.3. Total Page Frames

The total set of page frames used by the‘d’ processes is

$$Z(t) = [Z_1(t) \cup Z_2(t) \cup Z_3(t) \ldots\ldots\ldots\ldots \cup Z_d(t)] \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ (4)$$

$$Z(t) = \bigcup_{i=1}^{d} Z_i(t) \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ (5)$$

This mathematical notation represents that total set of page frames used by the‘d’ processes is the union of‘d’ resident sets where the resident sets range from i= 1 to d

2.6.4. Total Number of Pages Shared by Processes

Suppose that there are two processes P_i andP_j. It is assumed that $P_i \neq P_j$.Also suppose that $z_{ij}(t)$ represents the number of pages shared by processes P_i and P_j at time instant‘t’.

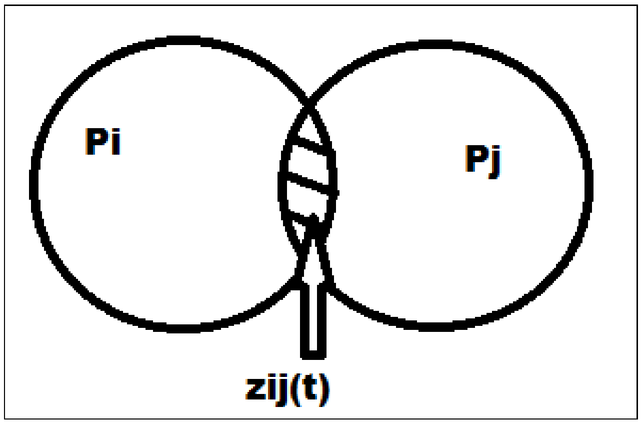

Figure 9. $z_{ij}(t)$ represent the number of pages shared by processes P_i and P_j

Suppose that there are three processes P_i, P_j, P_k and $P_i \neq P_j \neq P_k$. Also suppose that $z_{ijk}(t)$ represents the number of pages shared by processes P_i, P_j, P_k at time instant‘t’.

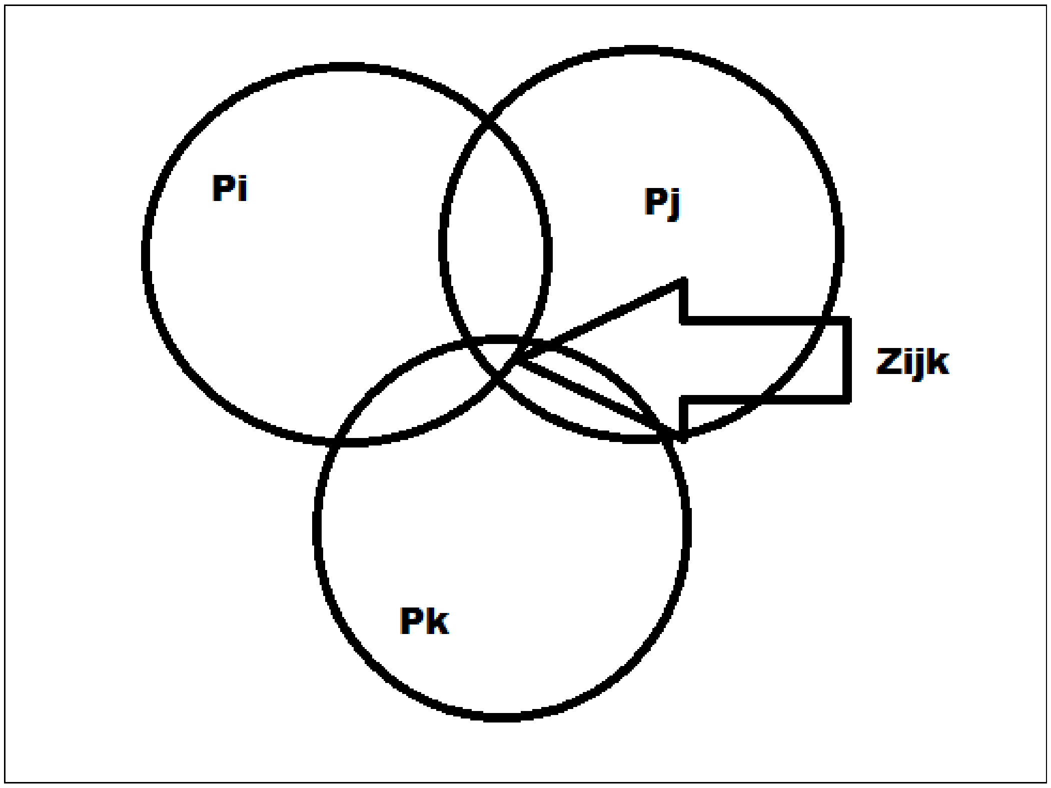

Figure 10. $z_{ijk}(t)$ represent the number of pages shared by processes P_i, P_j, P_k

When we include time instants t, we may write

$z_i(t) = n[Z_i(t)]$ .... (6)

$z_{ij}(t) = n[Z_i(t) \cap Z_j(t)]$ .... (7)

$z_{ijk}(t) = n[Z_i(t) \cap Z_j(t) \cap Z_k(t)]$ .... (8)

When we ignore time instants t, we may write

$z_i = n[Z_i]$ .... (9)

$z_{ij} = n[Z_i \cap Z_j]$ .... (10)

$z_{ijk} = n[Z_i \cap Z_j \cap Z_k]$ .... (11)

Here the following membership functions are true:

$z_i(t) \in P_i$ .... (12)

$z_{ij}(t) \in P_i, P_j$ .... (13)

$z_{ijk}(t) \in P_i, P_j, P_k$ .... (14)

Here $n(Z)$ is the number of pages in set Z. The sum of all $z(t)s$ with r subscripts represent the total number of pages shared by r processes at time instant t. Therefore, we observe that when we calculate total number of page frames, union operation is implemented and when we calculate total number of pages shared by processes, intersection operation is implemented as discussed in [18].

Suppose that there are 'r' processes at time instant 't', then the total number of pages shared by 'r' processes at time instant 't' will be represented by the sum of all $z(t)s$ with 'r' subscripts. This will be denoted as $N_r(t)$. Therefore we can write the following equations:

$N_1(t) = \sum z_i(t)$ .... (15)

$N_2(t) = \sum z_{ij}(t)$ .... (16)

$N_3(t) = \sum z_{ijk}(t)$ .... (17)

Here the following inequality exists to find out the range of number of processes:

$1 \leq i < j < k < \cdots \ldots\ldots\ldots\ldots\ldots\ldots.. \leq d$ .... (18)

Here'd' represents the total number of processes. The following block diagram shows the general notation for the sum of all $z(t)s$ with r subscript.

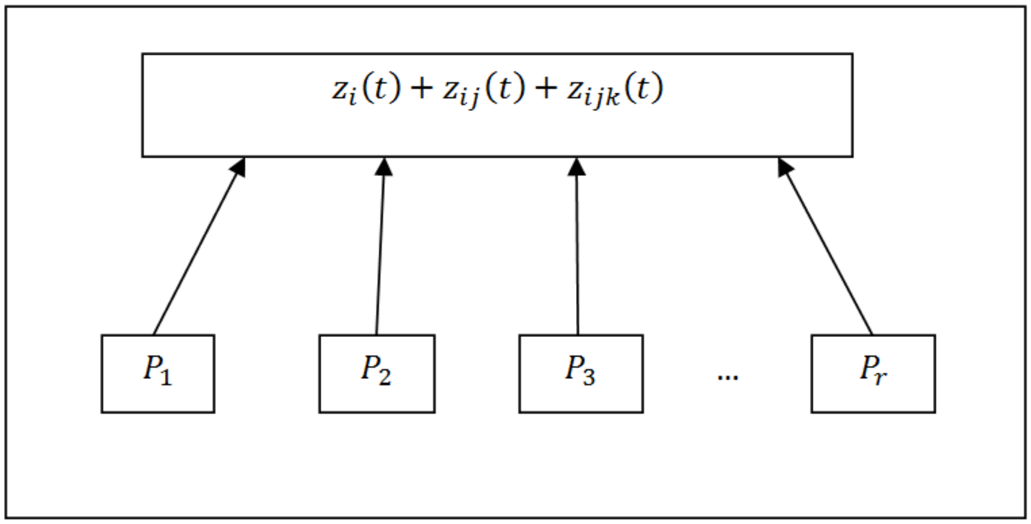

Figure 11. Total number of pages shared by 'r' processes at time instant't'.

When we consider the values of subscripts as follows:

$$i=1, j=2, k=3 \qquad \text{.....................} (19)$$

Then the above block diagram will become:

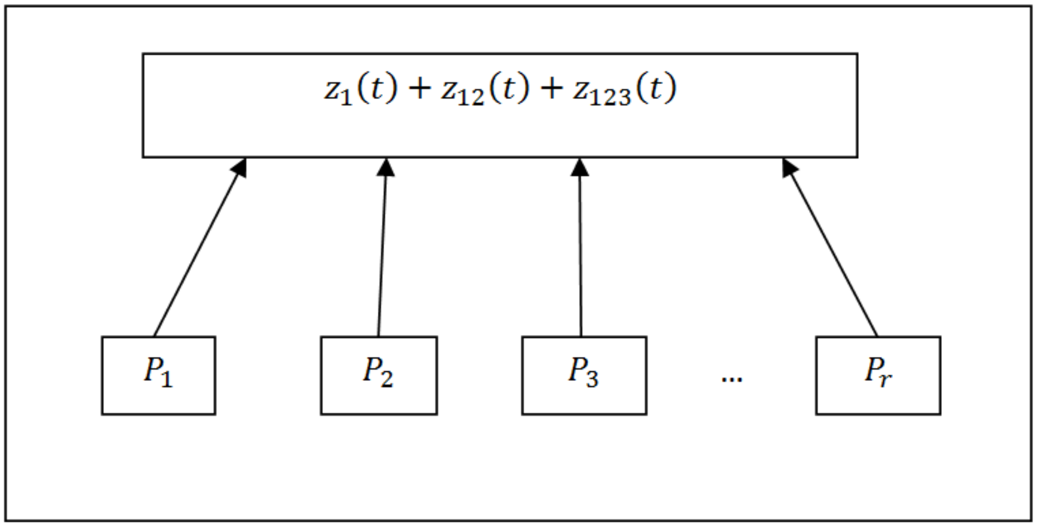

Figure 12. Total number of pages shared by 'r' processes at time instant't' for a case of $i=1, j=2, k=3$

2.7. Description of Feller's Equation or Inequality

The following subsections will describe the mathematical notation for the Feller's Equation as described in [22].

2.7.1. Total Number of Terms in $N_r(t)$

The term $N_r(t)$ has $\binom{d}{r}$ terms and last sum $N_d(t)$ reduces to a single term that indicates the number of pages shared by all the $'d'$ processes. The term $N_r(t)$ represents total number of pages shared by $'r'$ processes at time instant $'t'$.

2.7.2. Total Number of Terms in $N_d(t)$

The term $N_d(t)$ consists of only single term. This single term represents the number of pages shared by all the $'d'$ processes.

2.7.3. Feller's Equation or Inequality

The Feller's Equation or Inequality states that:

$$\sum_{r=1}^{d}(-1)^{r+1}N_r(t) \leq M \qquad \text{.............................. (20)}$$

In the Feller's equation we have:

$N_r(t)$: Total number of pages shared by $'r'$ processes at time instant $'t'$.
M: Total number of page frames available for allocation in memory.

We can observe that if we expand the Feller's equation we get:

$$\begin{aligned}
&(-1)^{1+1}N_1(t) + (-1)^{2+1}N_2(t) + (-1)^{3+1}N_3(t) + \cdots \leq M \\
&(-1)^{2}N_1(t) + (-1)^{3}N_2(t) + (-1)^{4}N_3(t) + \cdots \leq M \\
&N_1(t) - N_2(t) + N_3(t) - N_4(t) + \cdots \leq M \\
&\sum z_i(t) - \sum z_{ij}(t) + \sum z_{ijk}(t) - \sum z_{ijkl}(t) + \cdots \leq M
\end{aligned} \qquad \text{............... (21)}$$

Feller's equation states that either the total number of page frames available for allocation in memory will be equal to the total number of pages shared by $'r'$ processes at time instant $'t'$ or the total number of page frames available for allocation in memory will be greater than the total number of pages shared by $'r'$ processes at time instant $'t'$.Feller's equation can also be written as follows:

$$M \geq \sum_{r=1}^{d} (-1)^{r+1} N_r(t)$$ .... (22)

In equation A, we can notice that when we take the summation of total number of pages shared by $'r'$ processes at time instant $'t'$, we see that due to the term $(-1)^{r+1}$, the sign associated with $\sum z(t)$ continuously changes with alternative next term.

2.8. Resource Memory

In this survey, our purpose is to study and analyse the main memory with virtual memory and virtual memory allocation and management in computer systems. There are some pages in main memory which are used by active processes and there are certain pages in main memory which are not used by active processes as discussed in [16]. Those pages in main memory which are not used by active processes is called "Resource Memory". The Resource Memory is denoted by R(t) as follows:

$$R(t) = M - \sum_{r=1}^{d} (-1)^{r+1} N_r(t)$$ .... (23)

Since we are discussing the analytical techniques for the virtual memory allocation and management in computer systems, the complexity of analytical modelling techniques becomes more when we include the sharing concept.

For the sake of simplicity we assume that there is no sharing and Nr = 0 for r>1. This assumption simplifies the problem greatly and the equation of Resource Memory becomes:

$$R(t) = M - (-1)^{1+1} N_1(t) - (-1)^{2+1}.0$$ .... (24)

$$R(t) = M - N_1(t)$$ .... (25)

There is a specific method in memory management policy used to determine the program's set of localities. The purpose of the program's set of localities is to determine the contents and size of each process's resident set.

There are two types of partitioning: Fixed Partitioning and Variable Partitioning. In Fixed Partitioning, the partition vector or size vector does not varies with time't'. In Variable Partitioning, the partition vector or size vector varies with time't'.

Suppose that there is a process P_i and the process P_i is active in computer systems and the resident set size $z_i(t)$ is a fixed constant z_i for all time during which process P_i is active. In this case the Partition Vector or Size Vector is constant during which any set of active processes is fixed. This is known as Fixed Partition approach. In the Variable Partition approach, the partition vector or size vector is variable or the partition vector Z (t) varies with time. The partition vector or size vector Z (t) varies when any set of active processes change. There are some advantages of

Fixed Partitioning over Variable Partitioning but the Variable Partitioning is more advantageous than Fixed Partitioning.

The most important advantage of Fixed Partitioning is the low overhead of implementation. The reason for very low overhead of implementation is that during Fixed Partitioning, the partition vector changes very less frequently because we know that the partition vector changes if and only if the set of active processes change. Since the set of active processes change less frequently, therefore the partition vector Z (t) also change very less frequently.

Also the Fixed Partitioning approach has an advantage of large variance in locality set size. In Fixed Partitioning, the active processes change very less frequently. Even if the memory requirements for processes can be known prior to processing, the change in locality of processes occurs if there is a change in set of active processes.

We can correlate the change in locality with change in the set of active processes. Since in Fixed Partitioning, the change in partition vector Z (t) occurs less frequently, the set of active processes change less frequently, therefore there is a small change in locality. Suppose that we have a set of active processes $P_1, P_2, P_3, \dots\dots, P_d$ for Fixed Partitioning then each process P_i has a large variance in locality set size when time varies. If the time does not vary then each process P_i has zero variance in locality set size. In Fixed Partitioning, the computer system's page allocation algorithms suffer too much difficulty in page reallocation because the partition vector Z (t) is fixed. There is no way to reallocate page frames from Z_i to Z_j at a time whenP_i's locality is smaller than Z_i andP_j's locality is larger thanZ_j.

Suppose that the reallocation of page frames from Z_i to Z_j have advantages in terms of performance and reallocation of page frames of P_j would improve the performance and reallocation of page frames of P_i would not degrade the performance but due to the stringent requirements on partition vector size in fixed partitioning and fixed allocation of page frames to memory and active processes, reallocation of page frames is impossible even we view the advantages of reallocation of page frames. For the purpose of reallocation of page frames we have to move towards Variable Partitioning. After a deep study of Fixed Partitioning and Variable Partitioning, we see that the Variable Partitioning has far more advantages than Fixed Partitioning approach because there is a severe loss of memory utilization for processes that exhibit a wide variance of locality size as discussed in [22].

2.9. Memory Address Translation in the MIPS R2/3000

The MIPS R 2/3000 microprocessor employs an on-chip MMU as discussed in [21]. The MMU's primary function is to map 32-bit virtual addresses to 32-bit real addresses. Later members of the RX000 family like the R10000 support 64-bit addresses. A 32-bit address allows the R2/3000 to have a virtual address space of 2^{32} bytes or 4 GB. Both address spaces are composed of 4 KB pages , which are convenient block sizes for information transfer within a conventional memory hierarchy comprising a cache(of the split kind), main memory and secondary memory as discussed in [21]. The 4 GB virtual address space is further partitioned into four parts called "segments", three of which form the system region or "kernel region" in MIPS nomenclature , devoted to operating system functions, while the other is the user region, where application programs, data and control stacks are stored.

The format of an R2/3000 virtual address appears in the Figure 13, 14. It consists of a 20-bit virtual page address, referred to as the virtual page name VPN, and a 12-bit displacement D, which specifies the address of a byte within the virtual page. The high order 3-bits 31:29 of VPN form a type of tag that identifies the segment being addressed. Bit 31 of VPN is 0 for a user segment and 1 for a supervisor segment, it thus distinguishes the user and supervisor (privileged) control states of the CPU. The user segment is “kuseg” and occupies half the virtual address space. The supervisor region is divided into three segments, kseg0, kseg1, and kseg2, each of which has different characteristics as elaborated in [21].

Kuseg: This 2GB segment is designed to store all user code and data. Address in this region make full use of the cache and are mapped to real addresses via the TLB.

Kseg0: This 512 MB system segment is cached and unmapped: that is , virtual addresses within kseg0 are mapped directly into the first 512 MB of the real address space, which includes the cache, but no virtual address translation takes place. This segment typically stores active parts of the operating system.

Kseg1: This is also a 512 MB segment, but is both uncashed and unmapped. It is intended for such purposes as storing boot-up code which cannot be cached and for other instructions and data-high speed I/O data, for instance – that might seriously slow down cache operation.

Kseg2: This is a 1GB segment which, like kuseg, is both cached and mapped.

The MMU contains a TLB to provide fast virtual-to-real address translation. The TLB stores a 64-entry portion of the memory map (page table) assigned to each process by the operating system. The current virtual page addressVPN is used to access a 64-bit entry in the TLB which contains among other items, a 20-bit page frame number PFN. This real page address is fetched from the TLB and appended to the displacement D to obtain the desired 32-bit real address. An R2/3000 based system often has less than 4GB of physical memory, in which case not all the available real address combinations are used.

Observe that the VPN itself is also part of the TLB entry because a fast access method called “Associative Addressing” is used. Another major item stored in each TLB entry is the 6-bit process identification field PID. This field distinguishes each active program or process; hence up to 64 processes can share the available virtual page numbers without interference. There are also 4 control bits denoted NDVG, which define the types of memory accesses permitted for the corresponding TLB entry. For example N denotes non-cacheable when set to 1, it causes the CPU to go directly to main memory, instead of first accessing the cache. D is a write-protection (read-only) bit; an attempt to write when D=0 causes a CPU interrupt or trap.

The MMU has some features not shown in Figure 14, which are designed to trap error conditions that are collectively referred to as “Address Translation Exceptions”. When a trap occurs, relevant information about the exception is stored in MMU registers, which can be examined and modified by certain privileged instructions as discussed in [21]. A common address translation exception is a TLB miss, which occurs when there is no valid entry in the TLB that matches the current VPN. The operating system responds to a TLB miss by accessing the current process’s page table, which is stored in a known location in kseg2, and copying the missing entry to the TLB. Another address-translation exception type is an illegal access-for instance, a write operation addressed to a page with D=0(read only) in its TLB entry.

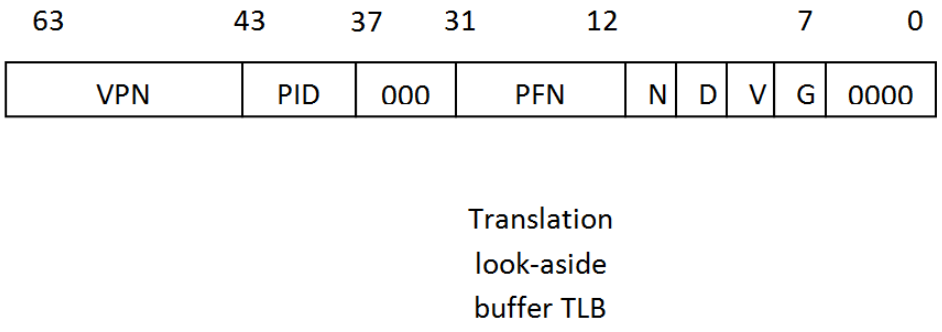

Figure 13. Translation look-aside buffer TLB in the MMU of MIPS R2/3000

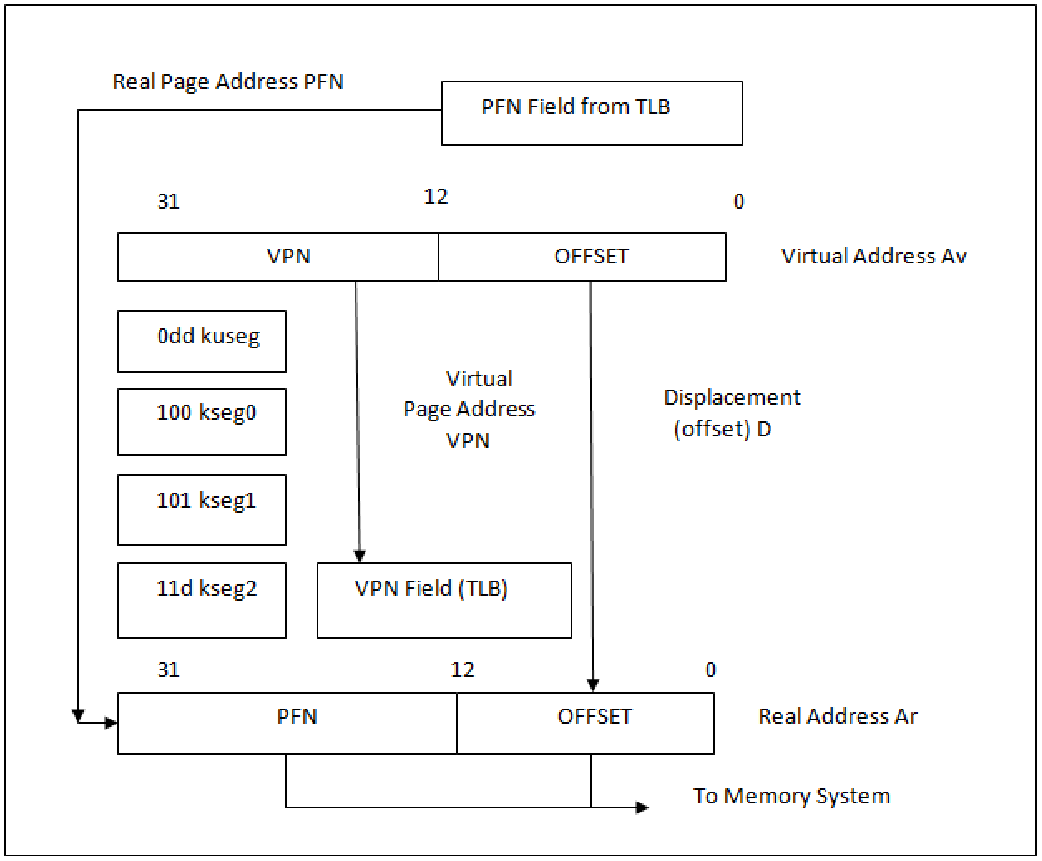

Figure 14. Memory address mapping in the MMU of the MIPS R2/3000

3. CONCLUSIONS

This paper is in fact a survey paper in which I have reviewed the Memory Management and Allocation in Computer Systems. I have started the review by elaborating the most fundamental concepts that is Queues in Computer Systems, type of Queues which are Long Term Queues, Short Term Queues, Intermediate Queues, I/O queues with their role in process scheduling in accordance with the readiness of process execution and Swapping. After discussing these fundamental concepts I got the conclusion that Swapping improves the performance of Computer Systems and the Virtual Memory improves the performance over swapping of Computer systems. In fact there is a very strong relation between “Memory Allocation and Management policy” and “System Load and System Parameters”. If there is a change in Memory Allocation and Management without changing the System Load and System Parameters then the overall performance of computer systems is improved. Also the Page Fault rate is improved .In this paper I have discussed the mathematical notation for Multiprogramming Systems. Also I developed the graphical models for overlapping of resident sets and active processes, the number of pages shared by two processes , the number of pages shared by three processes and total number of pages shared by ‘r’ processes at time instant ‘t’ using Set Theory. It is concluded that when we calculate total page frames, the union operation is implemented and when we calculate total number of pages shared by processes, the intersection operation is implemented. We have put light on the concept of Resource Memory in computer systems and we also discussed MIPS R2/3000 virtual to real address mapping in detail.

ACKNOWLEDGEMENTS

I would like to thank the ex- Director General of Bahria University Karachi campus, Vice Admiral Khalid Amin HI(M) (Retired)and the Director of Bahria University Karachi Campus, Captain Mohsin H. Malik TI(M) PN for motivating me to be involved in research work in my field of interest. They have always persuaded the faculty to do research work for the sake of serving the humanity, science and engineering. Also I would like to thank Dr. Farah Naz Lakhani for motivating me to contribute towards research in the field of Computer Systems Engineering and Computing.

REFERENCES

[1] M. Tofte and J.-P. Talpin, "Region-based memory management," Information and computation, vol. 132, pp. 109-176, 1997.
[2] C. A. Waldspurger, "Memory resource management in VMware ESX server," ACM SIGOPS Operating Systems Review, vol. 36, pp. 181-194, 2002.
[3] J. C. Lau, S. C. Roy, D. L. Callaerts, and I. E. N. Vandeweerd, "Method and apparatus for allocation and management of shared memory with data in memory stored as multiple linked lists," ed: Google Patents, 1999.
[4] P. R. Wilson, M. S. Johnstone, M. Neely, and D. Boles, "Dynamic storage allocation: A survey and critical review," in Memory Management, ed: Springer, 1995, pp. 1-116.
[5] K. Moronaga and M. Watanabe, "Storage management system for memory card using memory allocation table," ed: Google Patents, 1993.
[6] D. F. Hooper, G. Wolrich, M. J. Adiletta, and W. R. Wheeler, "Method for memory allocation and management using push/pop apparatus," ed: Google Patents, 2003.
[7] D. Gay and A. Aiken, Memory management with explicit regions vol. 33: ACM, 1998.

[8] C. A. Waldspurger and W. E. Weihl, "Lottery scheduling: Flexible proportional-share resource management," in Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation, 1994, p. 1.
[9] T. J. Lehman and M. J. Carey, "A study of index structures for main memory database management systems," in Proc. VLDB, 1986.
[10] E. D. Berger, B. G. Zorn, and K. S. McKinley, "OOPSLA 2002: reconsidering custom memory allocation," ACM SIGPLAN Notices, vol. 48, pp. 46-57, 2013.
[11] O. Avissar, R. Barua, and D. Stewart, "An optimal memory allocation scheme for scratch-pad-based embedded systems," ACM Transactions on Embedded Computing Systems (TECS), vol. 1, pp. 6-26, 2002.
[12] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, et al., "Query processing, resource management, and approximation in a data stream management system," 2003.
[13] F. Catthoor, S. Wuytack, G. de Greef, F. Banica, L. Nachtergaele, and A. Vandecappelle, Custom memory management methodology: Exploration of memory organisation for embedded multimedia system design: Springer Science & Business Media, 2013.
[14] I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, and A. Roy, "A distributed resource management architecture that supports advance reservations and co-allocation," in Quality of Service, 1999. IWQoS'99. 1999 Seventh International Workshop on, 1999, pp. 27-36.
[15] I. Puaut, "Real-time performance of dynamic memory allocation algorithms," in Real-Time Systems, 2002. Proceedings. 14th Euromicro Conference on, 2002, pp. 41-49.
[16] P. Zhou, V. Pandey, J. Sundaresan, A. Raghuraman, Y. Zhou, and S. Kumar, "Dynamic tracking of page miss ratio curve for memory management," in ACM SIGOPS Operating Systems Review, 2004, pp. 177-188.
[17] D. J. McMahon and G. A. Buzsaki, "Dynamic memory allocation in a computer using a bit map index," ed: Google Patents, 1998.
[18] K. Harty and D. R. Cheriton, Application-controlled physical memory using external page-cache management vol. 27: ACM, 1992.
[19] A. Aiken, M. Fähndrich, and R. Levien, Better static memory management: Improving region-based analysis of higher-order languages vol. 30: ACM, 1995.
[20] M. B. Jacobson, J. W. Fordemwalt, D. L. Voigt, M. D. Nelson, H. Vazire, and R. Baird, "Memory systems with data storage redundancy management," ed: Google Patents, 1995.
[21] John P. Hayes (1998) *Computer Architecture and Organization*, WCB/McGraw- Hill Books, McGraw-Hill International Editions.
[22] Denning P.J. and Graham G.S, “Multi-programmed Memory Management”, Proc. IEEE, vol 63, June 1975, pp 924-939
[23] William Stalling (2004) *Computer Architecture and Organization*, WCB/McGraw- Hill Books, McGraw-Hill International Editions.

AUTHOR

Mr. Ahmed Faraz holds Bachelor of Engineering in Computer Systems and Masters of Engineering in Computer Systems from N.E.D University of Engineering and Technology, Karachi Pakistan .He has taught various core courses of computer science and engineering at undergraduate and postgraduate level at Sir Syed University, N.E.D University and Bahria University Karachi for more than ten years. His research interests include Artificial Intelligence, Data Mining, Parallel Processing, Computer Architecture and Organization, and Statistical Learning.