Assignment - 02

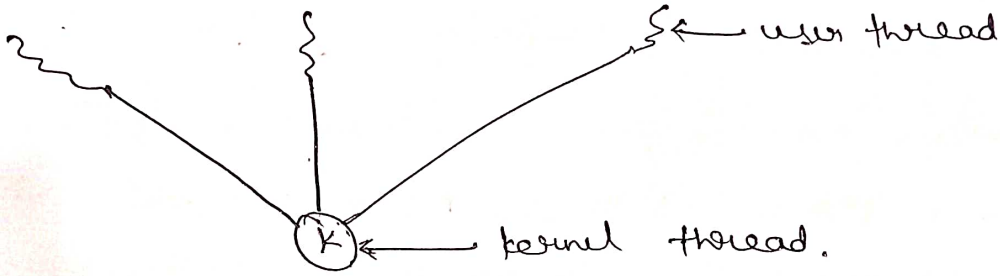
OS(18CS43)

Shahul Hameed. S
1K[redacted]18CS097
CSE 'A' Sec
4th Sem

(1) Explain multithreading models, Also list the benefits of multithreaded programming.

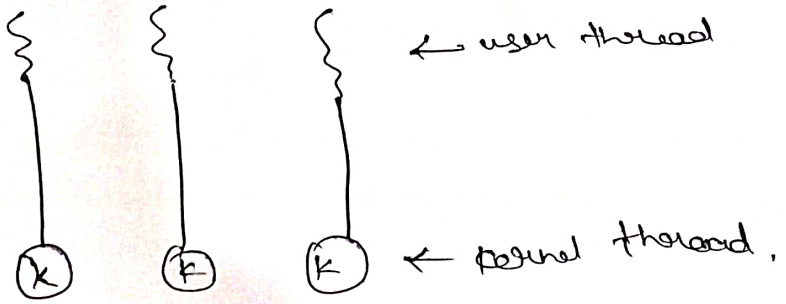
(a) Many-to-One :- Many user level threads mapped to single kernel thread.

Ex:- Solaris green threads, GNU portable threads.



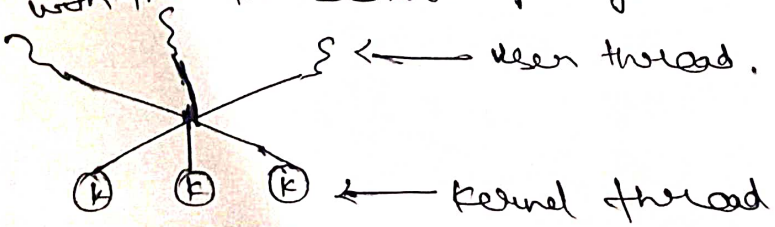
(b) One-to-One :- Each user-level thread maps to kernel thread

Ex:- Linux, Solaris 9 & later.



(c) Many-to-Many :- Allows many user level threads to be mapped to many kernel threads. Allows OS to create a different number of kernel threads.

Ex:- NT/2000 with the threadfiber package



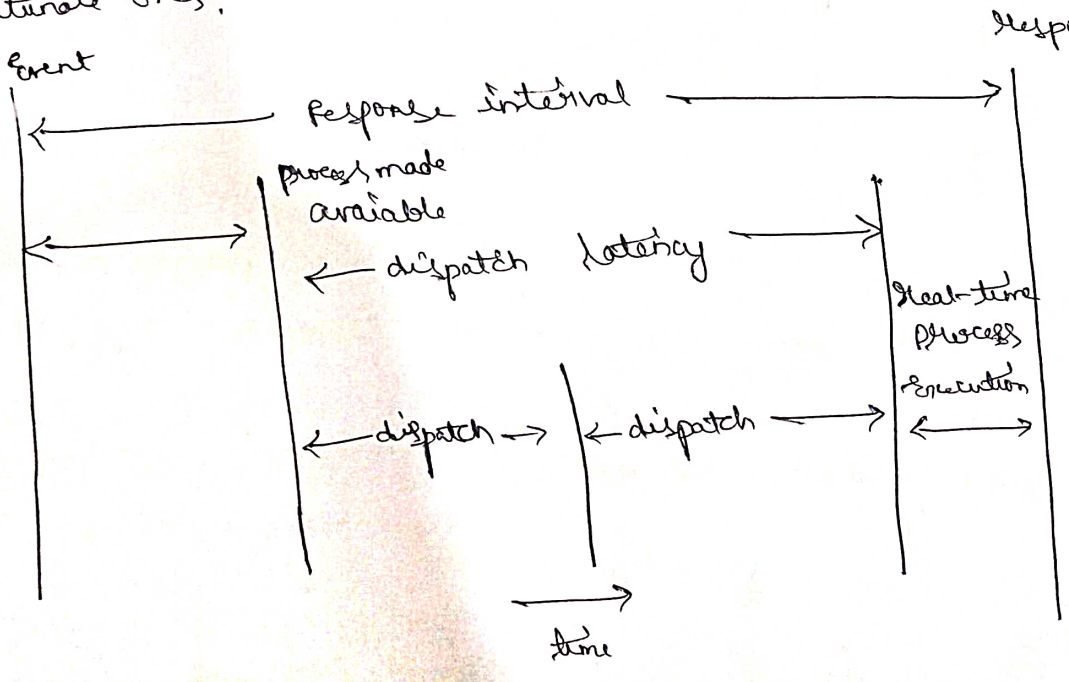
Benefits of Multiple thread programming:.

* A multiple thread makes a great Server for Example printer Server.
* It shares common data, they do not use inter-process communication.
* Responsiveness.
* Resource Sharing
* Economy & cheap.
* Utilization of MP Architectures.

(2) Explain Multiprocessor Scheduling.

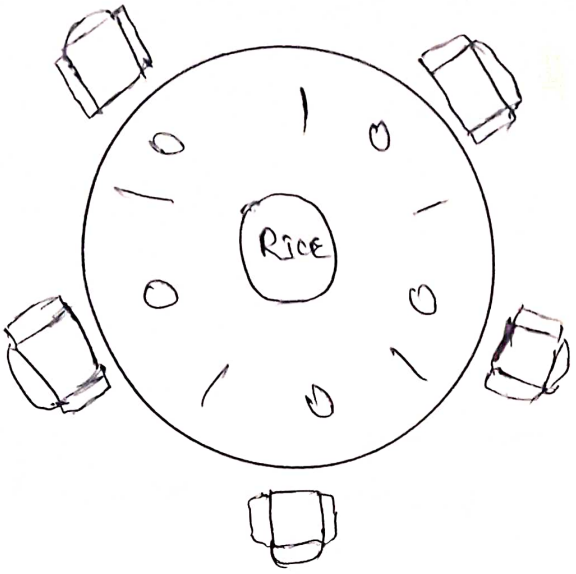
> CPU Scheduling more complex when multiple CPU is are available. Homogeneous process within a multiprocessor. Load Sharing, Asymmetric multiprocessing - only one processor accesses the system data structures. alleviating the need for data Sharing.

Hard real-time Systems - required to complete a Critical task within a guaranteed amount of time. Soft real time computing requires that critical processes receive priority over less fortunate ones.



(5). Explain Dining - philosophers problem with semaphores.

>



1. Shared data

Bowl of rice (data set)

Semaphore chopstick [5] initialized to

The structure of philosopher i: while(true) { wait (chopstick[i]);

wait (chopstick [(i+1) %.5]);

// Eat signal (chopstick[i]); signal (chopstick [(i+1)%.5]); // think

}

Problems with Semaphores:

Correct use of semaphore operations:

Signal (mutex) .... wait(mutex)

wait(mutex) .... wait(mutex)

Omitting of wait(mutex) or signal (mutex) (or both)

Solution to Dining philosophers

monitor DP

{ Enum {Thinking, (Hungry, Eating) state[5]; condition self[5];

void pickup(int i) { state[i] = Hungry; test(i); if (state[i] != Eating) self[i].wait;

}

void putdown (int i) { state[i] = Thinking;

```
test((i+4)%5); test((i+1)%5); }
void test(int i) { if ((state[(i+4)%5] != Eating) && (state[i] == Hungry)
&& (state[(i+1)%5] != Eating)) {
        state[i] = Eating; self[i].signal(); }}
initialization_code() { for (int i=0; i<5; i++) state[i] = Thinking;
}
}.
```

(4) Illustrate how Reader's - writer's problem can be solved by using semaphores.

A data set is shared among a number of concurrent processes

- Readers - only read the data set; they do not perform any updates.
- writers - Can both read & write.

problem: allow multiple readers to read at the same time. only one single writer can access the shared data at the same time.

Shared data,

- Data set
- Semaphore mutex initialize to 1.
- Semaphore wrt initialize to 1.
- Integer read count initialize to 0.

The structure of a writer process

```
while (true) { wait(wrt); // writing is performed
        signal(wrt);
        }
```

The structure of a read process

```
while (true){ wait(mutex); readcount ++;
    if(readcount == 1) wait(wrt); Signal(mutex)
        //reading is performed.
    wait(mutex);
    read count --;
    if(read count == 0) signal(wrt);
    Signal(mutex);
}
```

(5) what is paging? Explain paging hardware with translation look-aside buffer.

> Paging is a memory management scheme that permits the physical address space of a process to be non-contiguous. support for paging is handled by hardware. It is used to avoid external fragmentation.

The hardware implementation of the page table can be done in several ways:

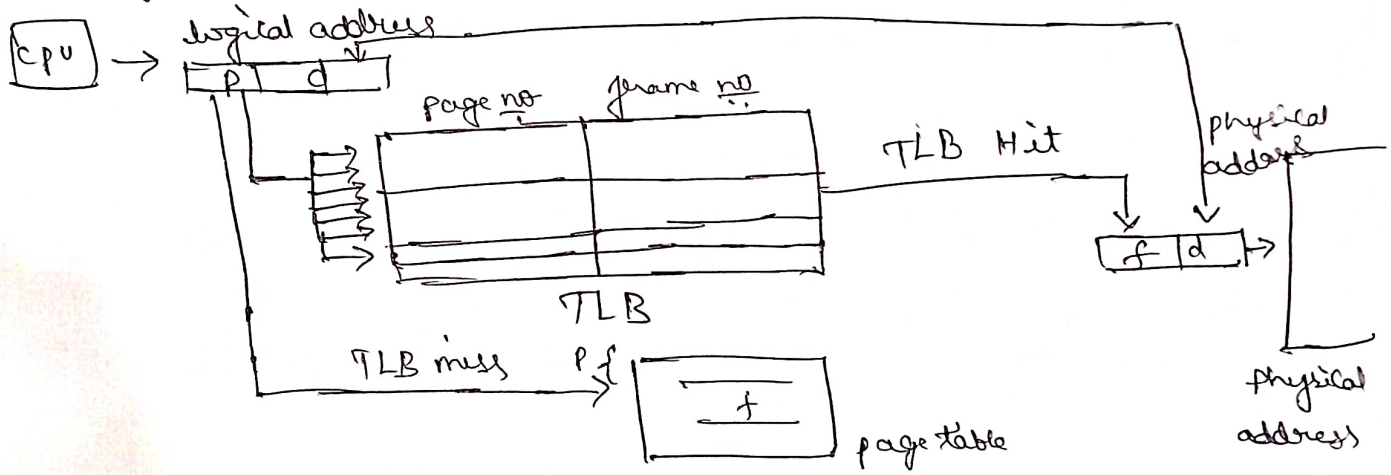
The simplest method is that the page table is implemented as a dedicated registers. These registers must be built with very high speed logic for making paging address translation. Every accessed memory must go through paging map. The use of registers for page table is satisfactory. If the page table is small.

If the page table is large than the use of registers is not visible so the page table is kept in the main memory and a page table base register [PTBR] points to the page table. changing the page table requires only one register which reduces the context switching type. The problem with this approach is the time

required to access memory location. to access a location[i] first we have to index the page table using PTBR offset. It gives the frame number which is combined with the ~~base~~ page offset produce the actual address. Thus we need two memory access for a byte.

The only solution is use special, fast, look up hardware cache called translation look aside buffer [TLB] or associative register.

TLB is built with associative register with high speed memory Each register contains two paths a key and a value.



when an associative register is presented with an item, it is compared with all the key values, if found the corresponding value field is returned and searching is fast.

TLB is used with the page table as follows: TLB contains only few page table entries. when a logical address is generated by the cpu, its page number along with the frame number is added to TLB. If the page number is found its frame memory is used to access the actual memory. If the page number is not in the TLB the memory reference to the page table is made. When the frame number is obtained we can use it to access the memory. If the TLB is full of entries the OS must select anyone for replacement. Each time a new page table is selected the TLB must be flushed to

Ensure that rest executing process do not use wrong information.
The percentage of time that a page number is found in the TLB is called Hit ratio.

(6) what are necessary conditions for deadlock? Explain different methods to recover from deadlock.

> A deadlock situation can occur if the following 4 conditions occur simultaneously in a system.

Mutual Exclusion:- only on process must hold the resources at a time. If any other process requests of the resource, the requesting process must be delayed until the resource has been released.

Hold and wait :- A process must be holding at least one resource & waiting to acquire additional resources that are currently being held by the other process.

Circular wait :- A set $\{P_0, P_1, \ldots, P_n\}$ of waiting process must exist such that P_0 is waiting for a resource i.e held by P_1. P_1 is waiting for a resource i.e held by P_2. P_{n-1} is waiting for resource held by process P_n and P_n is waiting for the resource i.e. held by P_1. All the four conditions must hold for a deadlock to occur.

No preemption :- Resources can be preempted i.e. only the process holding the resources must release it after the process has completed its task.

Deadlock prevention algorithm may lead to low device utilization and reduced system throughput. Avoiding deadlocks requires additional information about how resources are to be requested. with the knowledge of the complete sequence of requests & releases

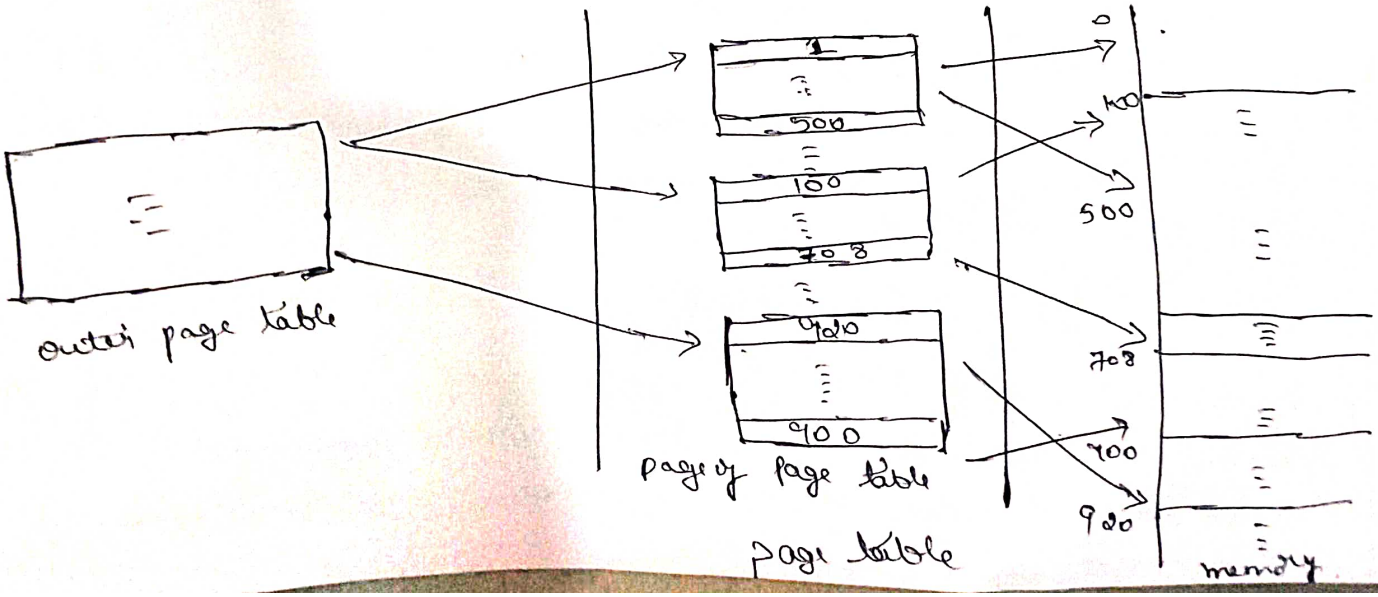
it can decide for each requests whether or not the process should wait. For each requests it requires to check the resources currently available, resources that are currently allocated to each processes future requests and release of each process to decide whether the current requests can be satisfied or must wait to avoid future possible deadlock.

(9) Explain the structure of page table.

7 a. Hierarchical paging:

Recent computer system support a large logical address space from 2^{32} to 2^{64}. In this system the page table becomes large. So it is very difficult to allocate contiguous main memory for page table. One simple solution to this problem is to divide page table in to smaller pieces.

One way is to use two-level paging algorithm in which the page table itself is also paged. Eg: In a 32 bit machine with page size of 4kb. A logical address is divided into a page number consisting of 20 bits and a page offset of 12 bit. The page table is further divided since the page table is paged. The page number is further divided into 10 bit page number and a 10 bit ~~offset~~ offset.



b. Hashed page table:

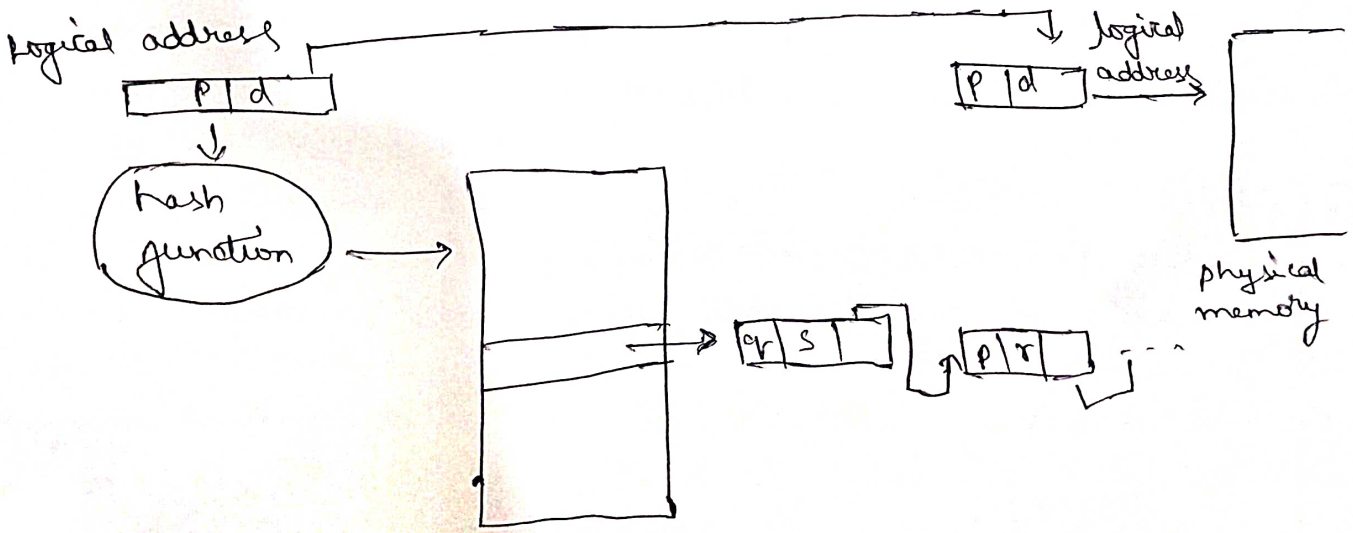
Hashed page table handles the address space larger than 32 bit. The virtual page number is used as hashed value. Linked list is used in the table which contains a list of elements that hash to the same location.

Each element in the hash table contains the following three fields. • virtual page number • mapped page number • pointer.

Working:

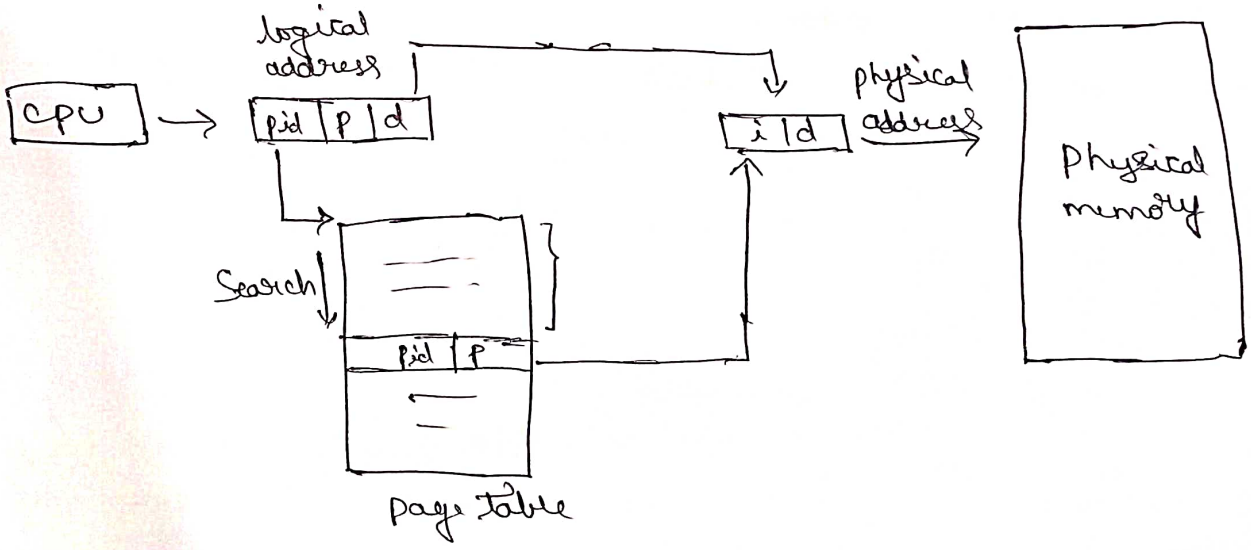
* virtual page number is taken from virtual address.
* virtual page number is hashed into hash table.
* virtual page number is compared with the first element of linking list.
* Both the values are matched, that value is used for calculating the physical address.

* If not match then entire linked list is searched for matching virtual page number.
* Clustered pages are similar to hash table but one difference is that each entity in the hash table refer to several pages.



c. Inverted page table: Since the address space have grown to 64 bits, the traditional page tables become a problem. Even with two level page tables. The table can be too large to handle. An inverted page table has only entry for each page in memory. Each entry consisted of virtual address of the page stored in that ready-only location with information about the process that owns that page.

Each virtual address in the inverted page table consists of triple <process-id, Page no., offset>. The inverted page table entry is a pair <process-id, page number>, ~~when a memory offset~~ when a memory reference is made, the part of virtual address, i.e <process-id, page-number> is presented into memory sub-system. The inverted page table is searched for a match. If a match is found at entry i then the physical address <i, offset> is generated.



(8) write a short note on:

(i) External & internal fragmentation!

(ii) Dynamic loading & linking

(i) External & internal fragmentation: In internal fragmentation there is wasted space internal to a portion due to the fact that block of data loaded is smaller than the partition.

Eg:- If there is a block of 50 Kb and if the process requests 40 Kb and if the block is allocated to the process then there will be 10Kb of memory left.

External fragmentation exists when there is enough memory space exists to satisfy the request, but it not contiguous; i.e. storage is fragmented into large number of small holes.

External fragmentation may be either minor or a major problem.

One solution for over coming external fragmentation is compaction. The goal is to move all the free memory together to form a large block. Compaction is not possible always. If the relocation is static and is done at load time then compaction is not possible. Compaction is possible if the relocation is dynamic and done at execution time.

(ii) Dynamic loading and linking:-

Dynamic loading:- For a process to be executed it should be loaded into the physical memory. The size of the process is limited to the size of the physical memory. Dynamic loading is used to obtain better memory utilization. In dynamic loading the routine or procedure will not be loaded until it is called. Whenever a routine is called, the calling routine first checks whether the called routine is already loaded or not.

Dynamic linking:- Some OS supports only the static linking. Here, only main program is loaded into the memory, if the main program requests a procedure, the procedure is loaded and the link is established at the time of references.

When "Stub" is executed it checks whether the routine is present in memory or not. If not it loads the routine.

This feature can be used to update libraries i.e library is replaced

by a new version and all the programs can make use of this library.

More than one version of the library can be loaded in memory at a time and each program uses its version of the library. only the program that are compiled with the new version are affected by the changes incorporated in it. other programs linked before new version is installed will continue using older libraries this type of system is called "shared library".