REG NO: 17BCE0581

DATE 06|11|2019

CPU scheduling exercise

1. Consider the following process arrival times, and run time requirements:

cess Name	Arrival Time	Running Time
A	0	3
B	1	5
C	3	2
D	9	2

For each scheduling algorithm, fill in the table with the process that is running on the CPU (for timeslice-based algorithms, assume a 1 unit timeslice). For RR and SRTF, assume that an arriving thread is run at the beginning of its arrival time, if the scheduling policy allows it. The turnaround time is defined as the time a process takes to complete after it arrives.

Time	FIFO	Round Robin	Shortest Remaining Time First
0	A	A	A
1	A	B	A
2	A	A	A
3	B	C	C
4	B	B	C
5	B	A	B
6	B	C	B
7	B	B	B
8	C	B	B
9	C	D	B
10	D	B	D
11	D	D	D
Avg TAT	(3+7+7+3)/4 = 5	(6+10+4+3)/4 = 5.75	(3+9+2+3)/4 = 4.25

2. Consider the following set of processes, with the length of the CPU burst given in milliseconds.

ocess	rst Time	rival Time	iority
P_1	10	~~8~~ 0	3
P_2	1	~~0~~ 0	1
P_3	2	~~0~~ 0	3
P_4	1	~~0~~ 0	4
P_5	5	~~2~~ 0	2

The processes are assumed to have arrived in the order P1, P2, P3, P4, P5 all at time 0.

a) Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, non-preemptive, priority (a smaller priority number implies a higher priority), and RR (quantum = 1).

b) What is the turnaround time of each process for each of the scheduling algorithms in part a?

c) What is the waiting time of each process for each of the scheduling algorithms in part a?

d) Fill the table for SRTF and RR

a)FCFS

P_1	P_2	P_3	P_4	P_5

0 10 11 13 14 19

a)SJF NP

P_2	P_4	P_3	P_5	P_1

0 1 2 4 9 19

a) Priority

P_2	P_5	P_1	P_3	

0 1 6 16 18 19

a) Round Robin

P_1	P_2	P_3	P_4	P_5	P_1	P_3	P_5	P_1	P_5	P_1	P_5	P_1	P_5	P_1	P_1	P_1	P_1	P_1

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

b) FCFS 13.4 SJF NP 7 Priority 13.4 RoundRobin 9.2

c) FCFS 9.6 SJF NP 3.2 Priority 8.2 RoundRobin 5.4

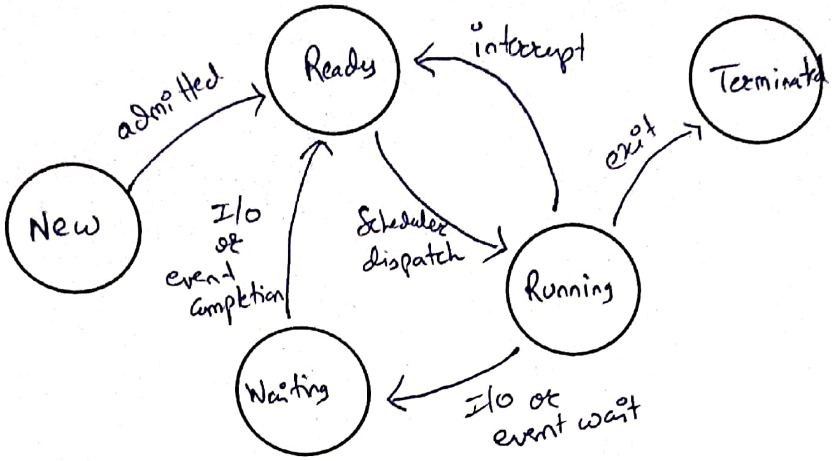
d) SRTF

ocess	rst Time	rival me	rting me	aiting me	ish Time	T
P_1	10	0	9	9	19	19
P_2	1	0	0	0	1	1
P_3	2	0	2	2	4	4
P_4	1	0	1	1	2	2
P_5	5	0	4	4	9	9
Avg				3.2		7

d) RR

ocess	rst Time	rival me	rting me	aiting me	ish Time	T
P_1	10	0	0	9	19	19
P_2	1	0	1	1	2	2
P_3	2	0	2	5	7	7
P_4	1	0	3	3	4	4
P_5	5	0	4	9	14	14
Avg				5.4		9.2

1. Complete for 5 state processes



2. Give three examples of an explicit hardware mechanism that is motivated by specific OS services.

- Traps and traps vectors for handling internal errors and system calls.
- Kernel / user mode, base / limit registers, protected instructions for various forms of protection
- Interrupts and memory mapped communication for I/O

3. What are the differences between user-level and kernel-level threads? Under what circumstances is one type better than the other? What is the essential cause of the difference in cost between a context switch for kernel-level threads and a switch that occurs between user-level threads?

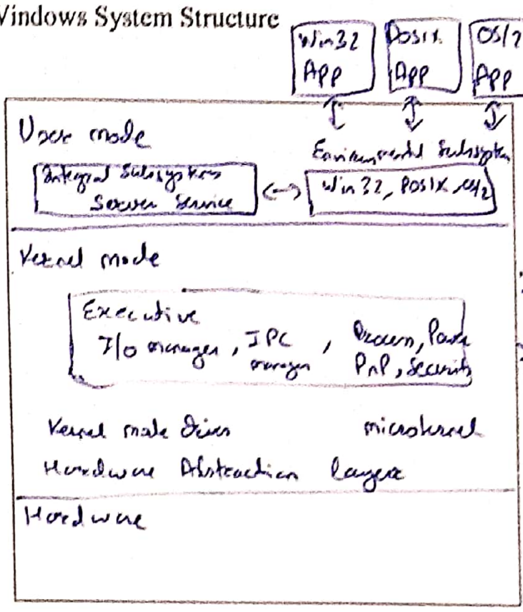
User level threads are threads that the OS is not aware of. They exist entirely within a process, and are scheduled to run within that process' timeline. The OS is aware of kernel level threads.
User level threads are much faster to switch b/w, as there is no context switch further a problem domain dependent algorithm can be used to schedule among them. Kernel level threads are scheduled by the OS, and each thread can ~~...~~ ~~...~~ be granted its own time slices by the scheduling algorithm. It require system call for the switch to occur; user-level threads do not.

4. Explain why system calls are needed to set up shared memory between two processes. Does sharing memory between multiple threads of the same process also require system calls to be set up?

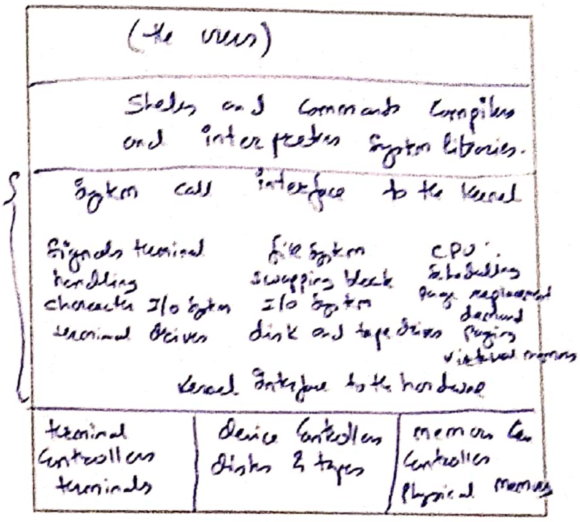
Because each process has its own address space, it needs to involve the kernel when dealing with other processes' address space. The kernel (not the process) has knowledge of the ~~...~~ physical memory mapping of all processes so it can determine a chuck of memory that can be used to share among multiple processes.

Threads do not need to use a system call to share memory because by definition, share their address space with other threads (of the same process, of course).

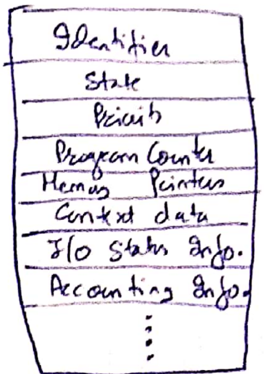
5. Draw Windows System Structure



UNIX system structure



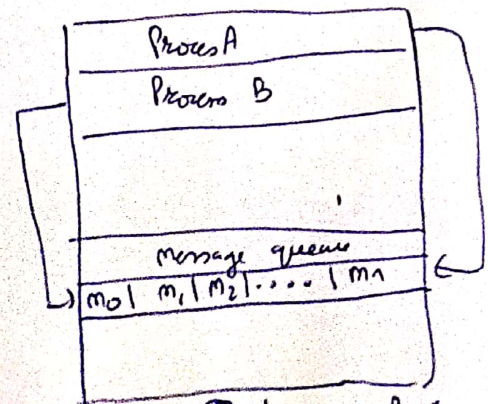
6. Draw Process Control Block and its switching



7. Communication models used in operating System.

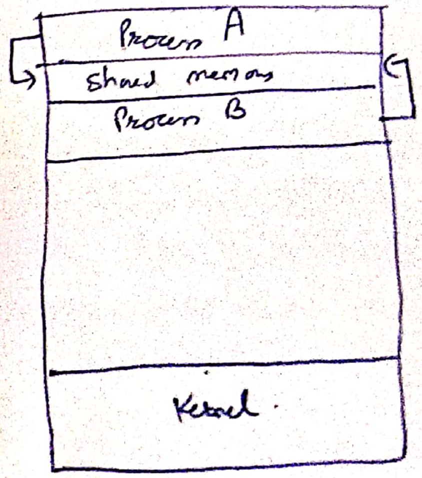
-> Shared Memory Model.

-> Message Passing Model



② Message Passing

① Shared Memory.



1. Solve the following

For the following problem, assume a hypothetical machine with 4 pages of physical memory and 7 pages of virtual memory. Given the access pattern:

A B C D E F C A A F F G A B G D F F

Indicate in the following table which pages are mapped to which physical pages for each of the following policies. Assume that a blank box matches the element to the left. We have given the FIFO policy as an example.

Access →		A	B	C	D	E	F	C	A	A	F	F	G	A	B	G	D	F	F
FIFO	1	A				E									B				
	2		B				F										D		
	3			C					A									F	
	4				D								G						
MIN	1	A															D		
	2		B														D		
	3			C									G				D		
	4				D	E	F												
LRU	1	A				F							G						
	2		B				F										D		
	3			C											B				
	4				D			A										F	

2. Solve for banker algorithm

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	3	3	2
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	2			

Need

	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	0

m=3, n=5 — Step 1 of Safety Algo

Work = Available

Work = | 3 | 3 | 2 |

	0	1	2	3	4
Finish =	false	false	false	false	false

For i = 0 — Step 2

Finish [0] = FALSE

For i = 1 — Step 2

Need$_1$ = [1,2,2]

Finish[1] = TRUE

Work = [3,3,2] + [2,0,0] — Step 3

Work = [5,3,2]

	0	1	2	3	4
Finish =	F	T	F	F	F

For i = 2 — Step 2

Need$_2$ = [6,0,0]

Finish[2] = FALSE

For i=3 — Step 2

Need$_3$ = [0,1,1]

Finish [3] = TRUE

Work = [5,3,2] + [2,1,1] — Step 3

Work = [7,4,3]

	0	1	2	3	4
Finish =	F	T	F	T	F

For i = 4 — Step 2

Need$_4$ = [4,3,0]

Finish [4] = TRUE

Work = [7,4,3] + [0,0,2] — Step 3

Work = [7,4,5]

	0	1	2	3	4
Finish =	F	T	F	T	T

For i = 0 — Step 2

Need$_0$ = [7,4,3]

Finish [0] = TRUE

Work = [7,4,5] + [0,1,0] — Step 3

Work = [7,5,5]

	0	1	2	3	4
Finish =	T	T	F	T	T

For i = 2 — Step 2

Need$_2$ = [6,0,0]

Finish [2] TRUE

Work = [7,5,5] + [3,0,2] — Step 3

Work = [10,5,7]

	0	1	2	3	4
Finish =	T	T	T	T	T

Finish [i] = true for $0 \le i \le n$ — Step 4

Hence the system is in Safe state

The safe sequence is

P_1, P_3, P_4, P_0, P_2

Scanned by CamScanner

3. Fill the structure of process 1 in first, second and third attempt and structure of process o in fourth attempt

```
/* PROCESS 0 /*
•
•
while (turn != 0)
    /* do nothing */ ;
/* critical section*/;
turn = 1;
•
```

```
/* PROCESS 1 */
•
•
while (turn != 1)
    /* do nothing */;
/* critical
   section */
turn = 0;
:
```

(a) First attempt

```
/* PROCESS 0 */
•
•
while (flag[1])
    /* do nothing */;
flag[0] = true;
/*critical section*/;
flag[0] = false;
•
```

```
/* PROCESS 1 */
•
•
while (flag[0])
    /* do nothing */
flag[1] = true;
  /* critical section */
flag[1] = false;
```

(b) Second attempt

```
/* PROCESS 0 */
•
•
flag[0] = true;
while (flag[1])
    /* do nothing */;
/* critical section*/;
flag[0] = false;
•
```

```
/* PROCESS 1 */
•
•
flag[1] = true;
while (flag[0])
    /* do nothing */
/* critical section */
flag[1] = false;
•
```

(c) Third attempt

```
/* PROCESS 0 */
:
flag[0] = true;
while (flag[1])
{ flag[0] = false;
  /* delay */;
  flag[0] = true;
}
/* critical section */
flag[0] = false;
•
```

```
/* PROCESS 1 */
•
•
flag[1] = true;
while (flag[0])
{
    flag[1] = false;
    /*delay */;
    flag[1] = true;
}
/* critical section*/;
flag[1] = false;
•
```

(d) Fourth attempt

4. There is one barber in the barber shop, one barber chair and n chairs for waiting customers. If there are no customers, the barber sits down in the barber chair and takes a nap. An arriving customer must wake the barber. Subsequent arriving customers take a waiting chair if any are empty or leave if all chairs are full. This problem addresses race conditions.
This solution uses three semaphores, one for customers (counts waiting customers), one for the barber (idle - 0 or busy - 1) and a mutual exclusion semaphore, mutex. When the barber arrives for work, the barber procedure is executed blocking the barber on the customer semaphore until a customer arrives. When a customer arrives, the customer procedure is executed which begins by acquiring mutex to enter a critical region. Subsequent arriving customers have to wait until the first customer has released mutex. After acquiring mutex, a customer checks to see if the number of waiting customers is less than the number of chairs. If not, mutex is released and the customer leaves without a haircut. If there is an available chair, the waiting counter is incremented, the barber is awaken, the customer releases mutex, the barber grabs mutex, and begins the haircut. Once the customer's hair is cut, the customer leaves. The barber then checks to see if there is another customer. If not, the barber takes a nap.

```
//Initial Condition
semaphore customers = 0;    semaphore barbers=0;
semaphore mutex=1;          int waiting=0;
```

```
Barber(){
  while (TRUE) {
     P(customers) ; /* go to sleep if no
                          customers */
     P(mutex);
     waiting = waiting - 1;
     V(barbers);
     V(mutex);
     cut-hair();
  }
}
```

```
Consumer(){
  P(mutex);
  if (waiting less than CHAIRS)
  {
     waiting = waiting + 1;
     V (customers);
     V (mutex);
     P(barbers);
     get-hair cut();
  }
  else {
     V(mutex);
  }
}
```