

REG NO:

DATE _____

CPU scheduling exercise

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

Process Name	Arrival Time	Running Time
	0	3
	1	5
	3	2
	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			
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
Avg TAT			

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

Process	First Time	Arrival Time	Priority

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

- 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).
- What is the turnaround time of each process for each of the scheduling algorithms in part a?
- What is the waiting time of each process for each of the scheduling algorithms in part a?
- Fill the table for SRTF and RR

--

--

--

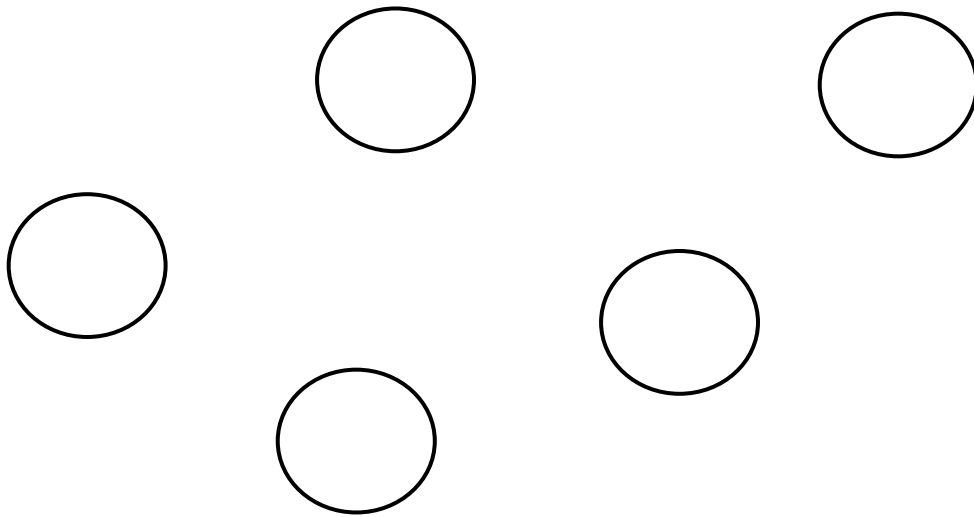
--

b) FCFS_____ SJF NP_____ Priority_____ RoundRobin_____

c) FCFS_____SJF NP_____Priority_____RoundRobin_____

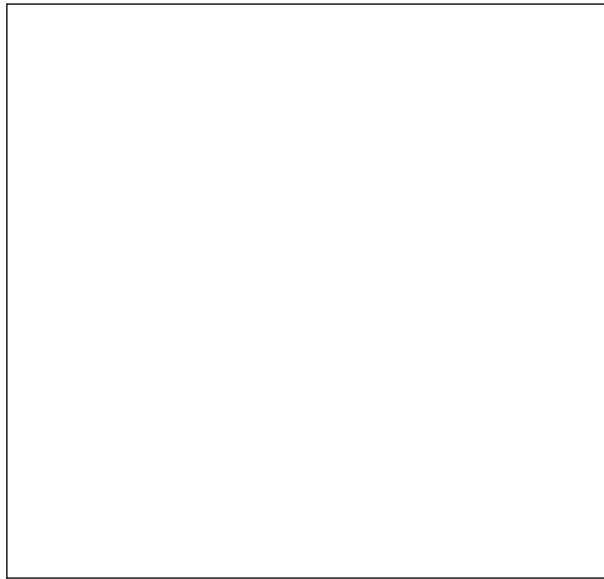
[illegible][illegible]

1. Complete for 5 state processes

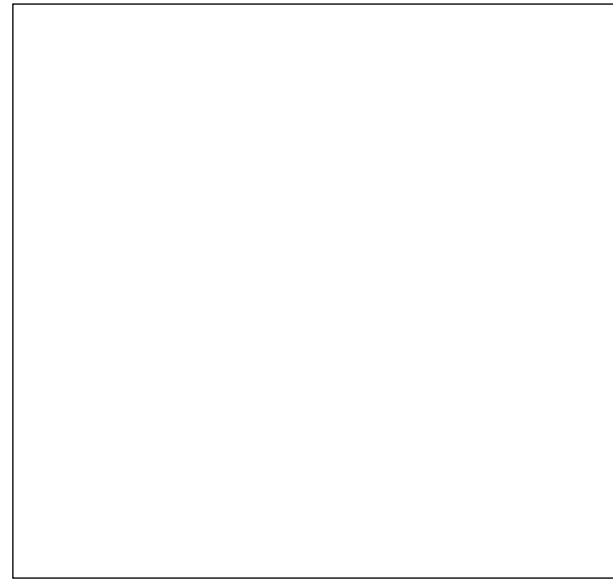


2. Give three examples of an explicit hardware mechanism that is motivated by specific OS services.
3. What are the differences between user-level and kernel-level threads? Under what circumstances is one type better than the other? What is the essential cause of the difference in cost between a context switch for kernel-level threads and a switch that occurs between user-level threads?
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?

5. Draw Windows System Structure



UNIX system structure



6. Draw Process Control Block and its switching

7. Communication models used in operating System.

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																		
	2																		
	3																		
	4																		
LRU	1																		
	2																		
	3																		
	4																		

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			

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] =

For i = 1 Step 2
 Need₁ =
 Finish[1] =

Step 3
 Work =
 Work =
 Finish =

Step 2
 For i = 2
 Need₂ =
 Finish[2] =

Step 2
 For i = 3
 Need₃ =
 Finish [3] =

Step 3
 Work =
 Work =
 Finish =

Step 2
 For i = 4
 Need₄ =
 Finish [4] =

Step 3
 Work =
 Work =
 Finish =

Step 2
 For i = 0
 Need₀ =
 Finish [0]

Step 3
 Work =
 Work =
 Finish =

Step 2
 For i = 2
 Need₂ =
 Finish [2]

Step 3
 Work =
 Work =
 Finish =

Step 4
 Finish [i] = true for 0 ≤ i ≤ n
 Hence the system is in Safe state

The safe sequence is

3. Fill the structure of process 1 in first, second and third attempt and structure of process 0 in fourth attempt

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

(a) First attempt

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

(b) Second attempt

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

(c) Third attempt

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

(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 awoken, 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(){

}

Consumer(){

}