CS343: Operating Systems
MID SEM, September 18, 2024
Indian Institute of Technology Guwahati

Duration: 2 hours **Total : 60 Marks**

Name: ________________________________ **Roll No.:** ______________

Student Signature: ________________________ **Invigilator Signature:** _____________

Q1. Answer the following questions. **[2+3+1+3+1+2= 12 Marks]**

a. State four events that may lead to process context switch in a time-sharing operating system.

Ans.

(i) A timer interrupt arrives
(ii) An I/O interrupt arrives

(iii) The process executes a syscall instruction

(iv) The process terminates execution

b. Consider the following assembly instructions:

save => saving registers
restore => restoring registers

The main and the fun subroutines have the following statements

main(){
add r1, r2
call fun()
}
	fun()
	{
	add r1, r2
	}

Place appropriately the save and restore instructions to perform caller saved and callee saved for both registers r1, and r2.

Caller save		**Callee save**	

main()
{
 	
}
	fun()
	{
	
	}
		main()
		{
		
		}
			fun()
			{
			
			}

Ans.

Caller save		**Callee save**	

**main(){
add r1, r2
save r1, r2
call fun()
restore r1, r2
}**
	**fun()
	{
	add r1, r2
	}**
		**main()
		{
		add r1, r2
		call fun()
		}**
			**fun(){
			save r1, r2
			add r1, r2
			restore r1, r2
			}**

c. Name any two events that can cause a trap in xv6.

Ans. Any two: System Calls, Interrupts, Program Faults

d. In xv6, consider "eip" pointing to user program, "esp" to user stack, suppose an interrupt occurs now, what instruction is responsible to handle the interrupt and what are the steps it performs?

Ans. The “int n” instruction
The following steps are performed by CPU as part of “int n” instruction

- **Fetch n-th entry interrupt descriptor table (CPU knows memory address of IDT)**
- **Save stack pointer (esp) to internal register**
- **Switch esp to kernel stack of process (CPU knows location of kernel stack of current process)**
- **On kernel stack, save old esp, eip (where execution stopped before interrupt occurred, so that it can be resumed later)**
- **Load new eip from IDT, points to kernel trap handler**

e. What is the assembly function responsible for pushing all the general-purpose registers during the kernel trap handler in xv6? What is the name of the C function it invokes?
Ans. alltraps and it invokes the “trap” function

f. In xv6, the timer interrupt ensures a process does not run for too long. What function does it invoke when a process (say process P1) exceeds its time? What happens to the state of the process P1, and what function does the above function call to give up its CPU? Name that function.
Ans. yield(), change the state of P1 from running to runnable, sched().

Q2. Answer the following questions. **[1+3+2+2+2= 10 Marks]**

a. In xv6, the scheduler thread is responsible for context switching, during context switching the following may occur.
   i. Switching from scheduler thread to a process
   ii. Switching from a process to a scheduler thread

   Name the functions responsible for case i and case ii, respectively.

   Ans. i) scheduler()   ii) sched()

b. In chronological order, list the activities that the “allocproc” function performs in xv6?

   Ans.

   1. **Find unused entry in ptable, mark is as embryo**
   2. **Marked as runnable after process creation completes**
   3. **New PID allocated**
   4. **New memory allocated for kernel stack**
   5. **Go to bottom of stack, leave space for trapframe**
   6. **Push return address of “trapret”**
   7. **Push context structure, with eip pointing to function “forkret”**

c. In xv6, briefly explain the difference between the trapframe and the context structure. (Hint: Explain w.r.t to how trapframe and context structure are saved.)
   Ans.
   - **Trapframe is saved when CPU switches to kernel mode (e.g., eip in trapframe is eip value where syscall was made in user code)**
   - **Context structure is saved when process switches to another process (e.g., eip value when swtch is called)**

d. In xv6, during context switching, briefly explain what the swtch() function does?
   Ans.
   - **Push remaining registers on old kernel stack (only callee save registers need to be saved)**
   - **Save pointer to this context into context structure pointer of old process**
   - **Switch esp from old kernel stack to new kernel stack**

- **ESP now points to saved context of new process**
- **Pop callee-save registers from new stack**
- **Return from function call (pops return address, caller save registers)**

e. In xv6, during the creation of new process (fork()), state whether the child process inherits from the parents or creates its own w.r.t the following.
   i. Memory image and file descriptors of the child process
   ii. Trapframe of the child process
   iii. Physical and logical memory of the child process
   iv. Return value in eax

   Ans.

   i. Parent memory and file descriptors copied
   ii. Trapframe of child copied from that of parent
   iii. Different physical memory but same virtual address (location in code)
   iv. Only return value in eax is changed, so parent and child have different return values from fork

Q3. Answer the following questions. **[3+2+6+4= 15 Marks]**

a. What are the four circumstances under which the CPU-scheduling decisions happen? What are the conditions of preemptive scheduling?

Ans.

1. **When a process switches from the running state to the waiting state.**
2. **When a process switches from the running state to the ready state**
3. **When a process switches from the waiting state to the ready state**
4. **When a process terminates**

Conditions 2 & 3 are the conditions of preemptive scheduling

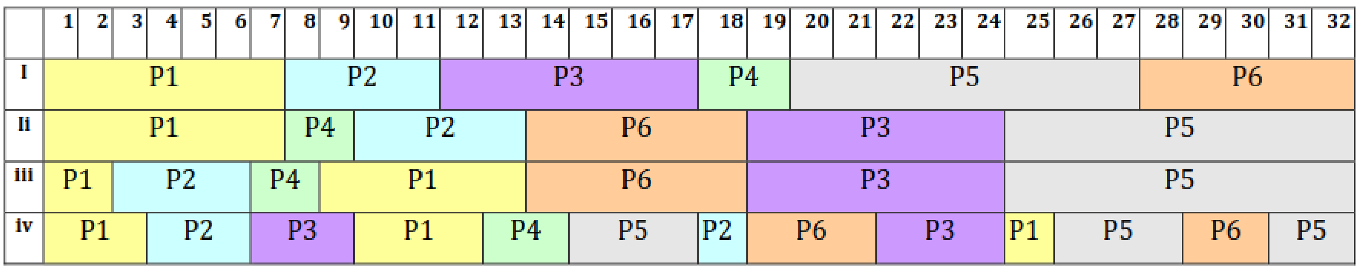
b. Justify with reasons whether the following CPU scheduling algorithms can result in process starvation: (i) first-come first-serve, (ii) shortest-job first.

   (i) The FCFS scheduling algorithm can lead to increased waiting time for processes if a long process starts executing, but there will be no starvation.
   (ii) The SJF algorithm on the other hand can lead to starvation, where a continuous stream of processes with shorter CPU bursts can prevent a process with longer CPU burst from getting the CPU.

c. Consider the following set of processes. Draw the Gantt chart for (i) FCFS, (ii) non-preemptive SJF, (iii) pre-emptive SJF (also known as SRTF), and (iv) round-robin with time quantum of 3 msec. Calculate the average waiting time and average turnaround time for all the above-mentioned scheduling algorithms.

Process	P1	P2	P3	P4	P5	P6
Arrival Time (ms)	0	2	3	5	6	8
CPU Burst (ms)	7	4	6	2	8	5

The Gantt chart for the four scheduling algorithms is shown below:



i) FCFS

AWT = (0 + 5 + 8 + 12 + 13 + 19) / 6 = 9.5
ATT = (7 + 9 + 14 + 14 + 21 + 24) / 6 = 14.83

(ii) Non-preemptive SJF

AWT = (0 + 7 + 15 + 2 + 18 + 5) / 6 = 7.83
ATT = (7 + 11 + 21 + 4 + 26 + 10) / 6 = 13.17

(iii) Preemptive SJF

AWT = (6 + 0 + 15 + 1 + 18 + 5) / 6 = 7.5
ATT = (13 + 4 + 21 + 3 + 26 + 10) / 6 = 12.83

(iv) Round-robin

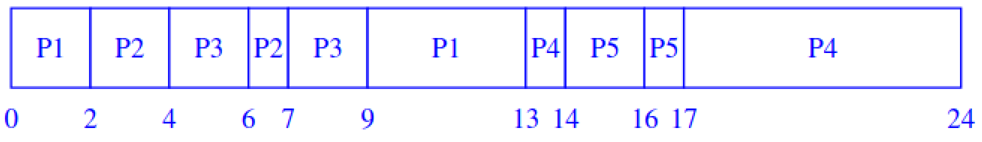
AWT = (18 + 12 + 15 + 7 + 18 + 17) / 6 = 14.5
ATT = (25 + 16 + 6 + 9 + 26 + 22) / 6 = 17.33

d. Consider a multi-level queue scheduling setup with one CPU and two ready queues Q0 and Q1. Q0 and Q1 are assigned to store the foreground processes and the background processes, respectively. The Round Robin scheduling is implemented for queue Q0 with time quantum = 2, whereas FCFS scheduling is implemented for queue Q1. Preemptive priority scheduling is followed across the two ready queues, with Q0 having the higher priority. Consider the five processes given in the following table. All times are in milliseconds (ms)

Process	P1	P2	P3	P4	P5
Arrival Time (ms)	0	2	3	4	14
CPU Burst (ms)	6	3	4	8	3
Type	Background	Foreground	Foreground	Background	Foreground

Draw the Gantt chart in this case and calculate the average waiting times of all the processes. There is no need to consider context-switch times in the Gantt chart.

Ans. The Gantt chart is given below



From this chart, we get the following waiting times.

P1: 13 – 0 – 6 = 7

P2: 7 – 2 – 3 = 2

P3: 9 – 3 – 4 = 2

P4: 24 – 4 – 8 = 12

P5: 17 – 14 – 3 = 0

Q4. Answer the following questions. **[1+2+3+3+3+3+3= 18 Marks]**

a. What does the pipe system call return, and what are they used for?
Ans.

Pipe system call returns two file descriptors. Data written in one file descriptor can be read through another

b. Define a critical section problem. Specify the requirements to be satisfied by the solution to the critical section problem.

Ans. If multiple processes have a shared section of code, in which the shared code should be executed by only one process at any point of time, then the problem of handling the shared code executed by only one process is know as critical section problem. Requirements to be satisfied by the solution to the critical section problem: (i) mutual exclusion, (ii) progress and (iii) bounded waiting.

c. Consider the following pseudo-code for a process **Pi**, where **"shared boolean flag[2]"** is a variable declared in shared memory, initialized as: **flag[0] = flag[1] = FALSE;**

```
P (int i) {                // i=0 for P0, i=1 for P1
while (TRUE) {
        flag[i] = TRUE;
        while (flag[(i+1) % 2) == TRUE);
                < Critical Section >
        flag[i] = FALSE;
                < Remainder Section >
        }
   }
```

Explain whether this code solves the critical section problem for two processes P0 and P1.

Ans. This solution may result in a deadlock between the two processes, where neither P0 nor P1 can proceed to enter their critical section.
This will happen when P0 makes flag[0] = TRUE, and then there is a context switch.
P1 makes flag[1] = TRUE. Now both P0 and P1 will be waiting indefinitely in the while loop before entry to the critical section.

d. Three concurrent processes P1, P2 and P3 are concurrently updating a shared variable xyz (with initial value of 100) as follows:

P1: xyz = xyz + 10;
P2: xyz = xyz – 20;
P3: xyz = xyz * 2;

Each instruction will translate into a sequence of machine-level instructions. For example,

```
P1:     LOAD R1,0(xyz)          // Load the value xyz to register R1; R1 = value of xyz
        ADD R1,R1,10            // R1 = R1 + 10
        STORE R1,0(xyz)         // xyz = R1
P2:     LOAD R2,0(xyz)
        SUB R2,R2,20
        STORE R2,0(xyz)
P3:     LOAD R3,0(xyz)
        MUL R3,R3,2
        STORE R3,0(xyz)
```

What will be the maximum and minimum values of xyz after execution of the three processes?

Ans. Inconsistency may occur when a process P loads xyz into the register, and then there is a context switch to another process that uses the previous value of xyz to update it. After P resumes, it uses the old value of xyz to carry out the updation.
Minimum Value: P2 loads xyz, then P1 & P3 finishes, then P2 resumes.
The final value of xyz will be 80.

Maximum Value: P1 finishes, P3 loads xyz, then P2 finishes, then P3 resumes.
The final value of xyz will be 220

e. Write the code for the test-and-set (T&S) and atomic exchange (EXCH) instructions. How can you implement a spinlock using the test-and-set instruction?

Ans.

```
T&S R1, lock                 ;R1 = Mem[lock] ⊕Mem[lock] = 1
EXCH R1, lock                ; T=Mem[lock] ⊕ Mem[lock] = R1 ⊕ R1=T

acquire:
        T&S R2, lock ; R2 = Mem[lock] ⊕Mem[lock]=1
        BNEZ R2, acquire ; if (R2!=0) goto acquire (lock was set)
        JR R31 ; return

release:
        SD R0, lock ; M[lock] =0
        JR R31 ; return
```

f. Write the code for load linked (LL) and store conditional (SC) instructions. Using LL and SC, implement the fetch-and-increment (F&I) atomic operation.

`

Ans.

```
LL R1, lock     ; R1=Mem[lock]

SC R2, lock     ; if (no other thread has written lock since LL)
                ; Mem[lock] = R2 ⊕R2 =1
                ; else R2=0

F&I:
        LL R2, lock             ; R2=Mem[lock]
        DADDI R3,R2,#1          ; R3=R2+1
        SC R3, lock             ; store conditional
        BEQZ R3, F&I            ; if SC failed (R3==0) retry
```

g. What is a semaphore? What are the standard operations that modify the semaphore value? Provide their code segments. How does a semaphore handle the critical section problem using entry and exit sections?

Ans.
Semaphore is a synchronization tool. It is an integer variable, except initialization it is used to access by two atomic operations wait() and signal().

Semaphore operations

```
wait(S) {
        while (S <= 0) ; no-op
        S−−;
}

signal(S) {
        S++;
}
```

Handling critical section by semaphore:

```
do {
        wait (S) ;
        critical section
        signal(S);
        remainder section
    } while (TRUE);
```

Q5. Answer the following **[1+1+1+1+1+1= 5 Marks]**

Consider the following snapshot of a system. P0, P1, P2, P3, P4 are the processes and A, B, C, D are the resource types. The value in the table indicates the number of instances of a specific resource (for example: 3 3 2 1 under the last column indicates that there are 3 A-type, 3 B-type, 2 C-type and 1 D-type resources are available after allocating the resources to all five processes). The numbers under allocation-column indicate that those number of resources are allocated to various processes mentioned in the first column. The numbers under Max-column indicate the maximum number of resources required by the processes. For example: in 1st row under allocation-column 2 0 0 1 indicate there are 2 A-type, 0 B-type, 0 C-type and 1 D-type resources are allocated to process P0. Whereas 4 2 1 2 under Max-column indicate that process P0's maximum requirement is 4 A-type, 2 B-type, 1 C-type and 2 D-type resources. Answer the following questions using Banker's algorithm by providing all intermediate steps:

Process	Allocation	Max	Available
	A B C D	A B C D	A B C D
P0	2 0 0 1	4 2 1 2	3 3 2 1
P1	3 1 2 1	5 2 5 2	
P2	2 1 0 3	2 3 1 6	
P3	1 3 1 2	1 4 2 4	
P4	1 4 3 2	3 6 6 5	

a. How many instances of resources are present in the system under each type of a resource?

A = 12; B = 12; C = 8; D = 10;

b. Compute the Need matrix for the given snapshot of a system.

P0 = 2 2 1 1
P1 = 2 1 3 1
P2 = 0 2 1 3
P3 = 0 1 1 2
P4 = 2 2 3 3

c. Verify whether the snapshot of the present system is in a safe state by demonstrating an order in which the processes may complete.

The given snapshot of the system is in safe state. With the available resources the processes will complete their execution with the following sequences: P0-P3-(any combination of P1, P2 and P4)

d. If a request from process P1 arrives for (1,1,0,0), can the request be granted immediately?

Yes, the request (1 1 0 0) may be granted to P1, and the system will remain in safe state. With the available resources the processes will complete their execution with the following sequences: P0-P3-(any combination of P1, P2 and P4)

e. If a request from process P4 arrives for (0,0,2,0), can the request be granted immediately?

No, the above said request (0 0 2 0) cannot be granted to P4. In case, if you grant the resources the system will go to un-safe state and all the processes will be in deadlock.