Computer Operating Systems

Midterm Exam Spring 2025

Department of Computer Science,
Hofstra University

Q1 Multiple-choice questions: enter your answer keys here

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
B	B	C	C	A	C	B	B	B	C
11	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
C	B	B	C	C	C	C	C	B	A

Q1. Multiple-choice. (20 pts)

1. What is the "kernel" in an operating system?
A) The hardware component managing memory
B) The one program running at all times on a computer
C) A user interface for applications
D) A type of application program
Answer: B

2. What is dual-mode operation in an operating system?
A) Running two operating systems simultaneously
B) Providing two modes: kernel mode and user mode
C) Allowing two users to access the same process
D) Switching between two CPUs dynamically
Answer: B

3. Which system call replaces the current process image with a new one?
a) `fork()`
b) `wait()`
c) `exec()`
d) `exit()`
Answer: C

4. What is the purpose of the `wait()` system call?
a) To create a new process
b) To wait for I/O operations to complete
c) To suspend the parent process until a child process terminates
d) To destroy a process
Answer: C

5. What is the difference between `wait()` and `waitpid()`?
a) `waitpid()` allows specifying which child process to wait for, while `wait()` does not.
b) `wait()` waits for all processes, while `waitpid()` waits only for threads.
c) Both are identical in functionality.
d) `waitpid()` suspends processes, while `wait()` terminates them.
Answer: A

6. In which state is a process when it is waiting for an I/O operation to complete?
a) READY
b) RUNNING
c) BLOCKED
d) TERMINATED
Answer: C

7. What is the primary advantage of threads over processes?
a) Threads have separate address spaces.
b) Threads are cheaper to create and manage than processes.
c) Threads cannot share resources like processes can.
d) Threads only exist in kernel mode.
Answer: B

8. What does multithreading allow in modern operating systems?
a) Multiple address spaces per thread
b) Concurrent execution within the same address space
c) Execution of only one thread at any time in a system
d) Elimination of kernel threads
Answer: B

9. In the Readers/Writers problem, why might writers starve?
A) Writers have higher priority
B) New readers continuously acquire the lock before writers
C) Semaphores are initialized incorrectly
D) Mutex locks are not used
Answer: B

10. Which synchronization primitive combines a mutex with condition variables?
A) Spinlock
B) Semaphore
C) Monitor
D) Ticket lock
Answer: C

Q1. Multiple-choice. (20 pts)

11. What happens when sem_wait() is called on a semaphore with value 0?
A) Returns immediately
B) Increments the value to +1
C) Blocks until sem_post() is called
D) Causes a segmentation fault
Answer: C

12. Which condition variable operation wakes all waiting threads?
A) pthread_cond_signal()
B) pthread_cond_broadcast()
C) pthread_cond_wait()
D) pthread_cond_init()
Answer: B

13. A counting semaphore initialized to N allows:
A) Only one thread to access a resource
B) Up to N threads to access a resource simultaneously
C) Threads to bypass mutex locks
D) Priority inversion to occur
Answer: B

14. Which condition is NOT a necessary condition for deadlock?
A) Mutual exclusion
B) Hold-and-wait
C) Starvation
D) Circular wait
Answer: C

15. In a Resource-Allocation Graph (RAG), a deadlock is certain if:
A) There is a cycle and each resource has multiple instances
B) There is no cycle
C) There is a cycle and all resources have single instances
D) A thread requests two resources simultaneously
Answer: C

16. In a Resource Allocation Graph (RAG) with a cycle and multi-instance resources:
A) Deadlock is certain
B) Deadlock is impossible
C) Deadlock is possible but not certain
D) Starvation must occur
Answer: C

17. In Round Robin scheduling, if there are 10 jobs in the ready queue and time quantum=10ms, what's the maximum wait time for any job?
A) 40ms
B) 80ms
C) 90ms
D) 100ms
Answer: C

18. Which scheduling algorithm requires prior knowledge of job execution times?
A) FCFS
B) RR
C) SJF
D) Multilevel Queue
Answer: C

19. In exponential averaging for burst prediction ($\tau_n = \alpha t_{n-1} + (1-\alpha)\tau_{n-1}$), what does $\alpha=1$ imply?
A) Only consider historical average
B) Only consider most recent burst
C) Equal weight to all bursts
D) No prediction capability
Answer: B

20. What percentage of CPU time is lost to context switching if quantum=100 ms and switch cost=1 ms?
A) 1%
B) 2%
C) 5%
D) 10%
Answer: A

Q2 Processes and Threads (20 pts)

- For these questions, assume there is no error, i.e., the return value of fork() is never negative. Assume round-robin scheduling algorithm like in Windows or Linux. (You need to provide the possible outputs and explain why. You do not need to draw the figures to show the parent child relationships.)

Q2 Processes and Threads (a) ANS

- What is the output of this program? If there may be multiple possible outputs, list ALL possible outputs, and explain why. Assume the virtual memory address of variable a in the initial process, &a = 0x12345678.
- ANS: Arbitrary interleavings between parent's and child's printout:
  - Parent: int a is 2 at 0x12345678 \n
  - Child: int a is 0 at 0x12345678 \n
- Explanation: Processes do not share the same memory space, so a = 1+1 = 2 in parent, and a = 1-1 = 0 in child. Fork copies the address space of the parent to the child, so both parent and child print the same memory address &a = 0x12345678. They both write to the same file descriptor STDOUT, since file descriptors are copied over to the new process.

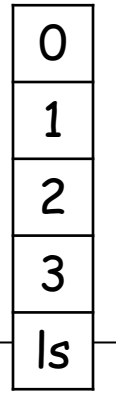
```
1 int main(void) {
2 int a = 1;
3 pid_t fork_ret = fork();
4 if (fork_ret > 0) {
5   a++;
6   fprintf(stdout, "Parent: int a is %d at %p\n", a, &a);
7 } else if (fork_ret == 0) {
8   a--;
9   fprintf(stdout, "Child: int a is %d at %p\n", a, &a);
10 } else {
11   printf("Fork error");
12 }
13 }
```

Q2 Processes and Threads (b) ANS

- For these questions, assume there is no error, i.e., all fork calls succeed, and the return value of fork() is never negative.
- 1. What does this program print?
- ANS: The program stops after printing 3, giving an output of
  - 0
  - 1
  - 2
  - 3
  - [Output of the 'ls' command, showing files in the current directory]
- Explanation: The program loops and prints the numbers 0, 1, 2, and 3. When i is 3, the execv function is called, which replaces the current process' memory and code with the /bin/ls program, and prints the current directory contents, then terminates.
- Explanation of execv("/bin/ls", argv);
  - The first argument ("/bin/ls") tells the operating system which executable file to load and run. This must be a valid path to the program.
  - The second argument (argv) is an array of strings that gets passed to the main function of the new program (ls in this case).
- argv[0] = "/bin/ls";
  - The very first element of the argv array (argv[0]) that a program receives is its own name or the path that was used to execute it.
  - When you run ls from your shell, the shell sets argv[0] to "ls" before executing the program. The ls program (and many others) uses argv[0] to know its own name, which is often used for printing error messages (e.g., "ls: cannot access 'no_such_file'").

```
1 int main(void) {
2 char** argv = (char**) malloc(3 * sizeof(char*));
3 argv[0] = "/bin/ls";
4 argv[1] = ".";
5 argv[2] = NULL;
6 for (int i = 0; i < 10; i++) {
7   printf("%d\n", i);
8   if (i == 3) {
9       execv("/bin/ls", argv);
10  }
11 }
12 return 0;
13 }
```

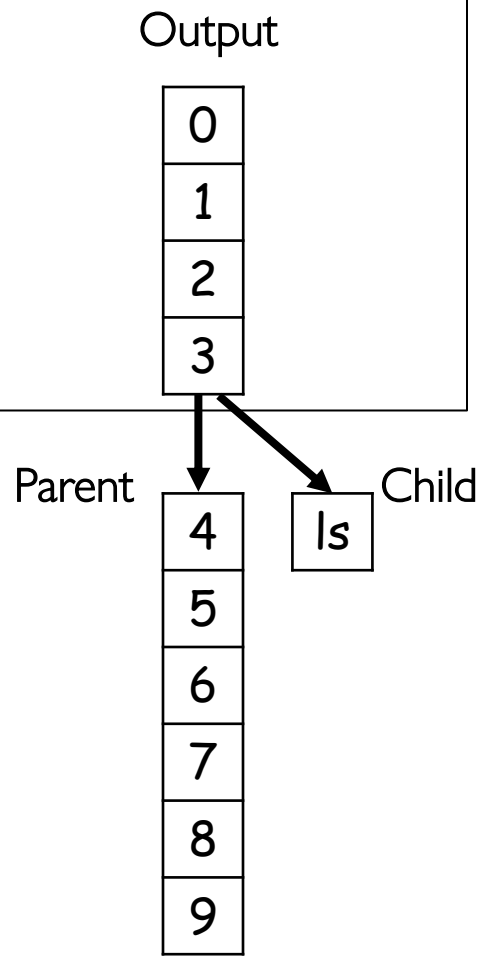
Output



Q2 Processes and Threads (c) ANS

- For these questions, assume there is no error, i.e., all fork calls succeed, and the return value of fork() is never negative. What does this program print?
- ANS: Parent prints 0 to 3, then calls fork() to create a child. Currently with arbitrary interleavings:
  - Parent prints 4 to 9.
  - Child prints the full output of ls command then terminates.

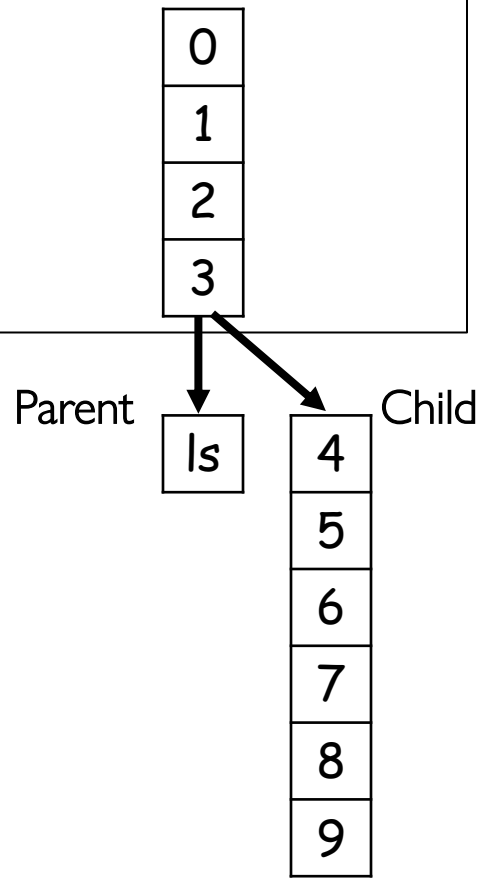
```
1 int main(void) {
2 char** argv = (char**) malloc(3 * sizeof(char*));
3 argv[0] = "/bin/ls";
4 argv[1] = ".";
5 argv[2] = NULL;
6 for (int i = 0; i < 10; i++) {
7  printf("%d\n", i);
8  if (i == 3) {
9      if(fork() == 0)
10         execv("/bin/ls", argv);
11 }
12 return 0;
13 }
```



Q2 Processes and Threads (d) ANS

- For these questions, assume there is no error, i.e., all fork calls succeed, and the return value of fork() is never negative. What does this program print?
- ANS: Parent prints 0 to 3, then calls fork() to create a child. Currently with arbitrary interleavings:
  - Parent prints the full output of ls command then terminates.
  - Child prints 4 to 9.

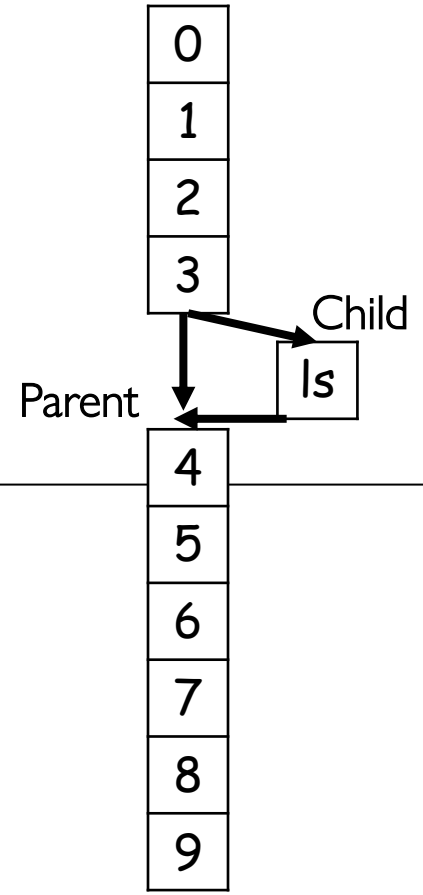
```
1 int main(void) {
2 char** argv = (char**) malloc(3 * sizeof(char*));
3 argv[0] = "/bin/ls";
4 argv[1] = ".";
5 argv[2] = NULL;
6 for (int i = 0; i < 10; i++) {
7  printf("%d\n", i);
8  if (i == 3) {
9      if(fork() > 0)
10         execv("/bin/ls", argv);
11 }
12 return 0;
13 }
```



Q2 Processes and Threads (e) ANS

- For these questions, assume there is no error, i.e., all fork calls succeed, and the return value of fork() is never negative. What does this program print?
- ANS: Parent prints 0 to 3, then calls fork() to create a child. Sequentially (with no interleaving)
  - Parent calls wait(NULL) and blocks.
  - Child executes execv and prints the output of the ls command, then terminates.
  - Parent resumes execution and continues its loop, printing 4 to 9

```
1 int main(void) {
2 char** argv = (char**) malloc(3 * sizeof(char*));
3 argv[0] = "/bin/ls";
4 argv[1] = ".";
5 argv[2] = NULL;
6 for (int i = 0; i < 10; i++) {
7  printf("%d\n", i);
8  if (i == 3) {
9       ret = fork();
10      if(ret == 0)
11          execv("/bin/ls", argv);
12      if(ret > 0)
13          wait(NULL);
14 }
12 return 0;
13 }
```



Q3 Synchronization (a) ANS

- (a) (5 pts) Consider the following concurrent program, where three threads access a shared variable x within critical sections protected by mutex locks. What are the possible final values of x after all threads finish execution? Explain why.
- ANS: 0, -1, and 1
- With mutex protection, no update to x will be erased. We consider all possible interleavings of the three threads.
  - Case 1: if T1 and T2 both run before T3, then x=0
  - Case 2: if T1 and T2 both run after T3, then x=0
  - Case 3: if T1 before T3 before T2, then x=-1
  - Case 4: if T2 before T3 before T1, then x=1

```
pthread_mutex_t mutex =
PTHREAD_MUTEX_INITIALIZER; //mutex is a
global shared mutex
int x=0; //x is a global shared variable
```

```
//Thread T1:
pthread_mutex_lock(&mutex);
x = x + 2;
pthread_mutex_unlock(&mutex);
```

```
//Thread T2:
pthread_mutex_lock(&mutex);
x = x - 2;
pthread_mutex_unlock(&mutex);
```

```
//Thread T3:
pthread_mutex_lock(&mutex);
x = x / 2;
pthread_mutex_unlock(&mutex);
```

Q3 Synchronization (b) ANS

- (b) (5 pts) Consider the following concurrent program, where three threads access a shared variable x without mutex locks. What are the possible final values of x after all threads finish execution? Explain why.
- ANS: -2,-1,0,1, or 2.
- Each update statement to x can be “erased” by “sneaking in between” the load and store of another update x statement. (c.f. Slide 5 “Race Condition” in L3 Synchronization.) The x=x+2 statement can either do nothing (if erased) or increase x by 2. The x=x-2 statement can either do nothing (if erased) or decrease x by 2. The x=x/2 statement can either do nothing (if erased) or divide x by 2.
- Case 1: none of the update statements are erased, so we have 3 possible outputs 0, -1, and 1 as in part a)
- Case 2: x=x+2 in T1 and x=x/2 in T3 are both erased, so only x=x-2 in T2 runs successfully, then x=-2. This is the minimum possible value of x.
- Case 2: x=x-2 in T2 and x=x/2 in T3 are both erased, so only x=x+2 in T1 runs successfully, then x=2. This is the maximum possible value of x.
- You may think of other possible cases, but it is not necessary to enumerate all of them, since they do not result in any new values for x.

```
int x=0; //x is a global shared variable
```

```
//Thread T1:
x = x + 2;
```

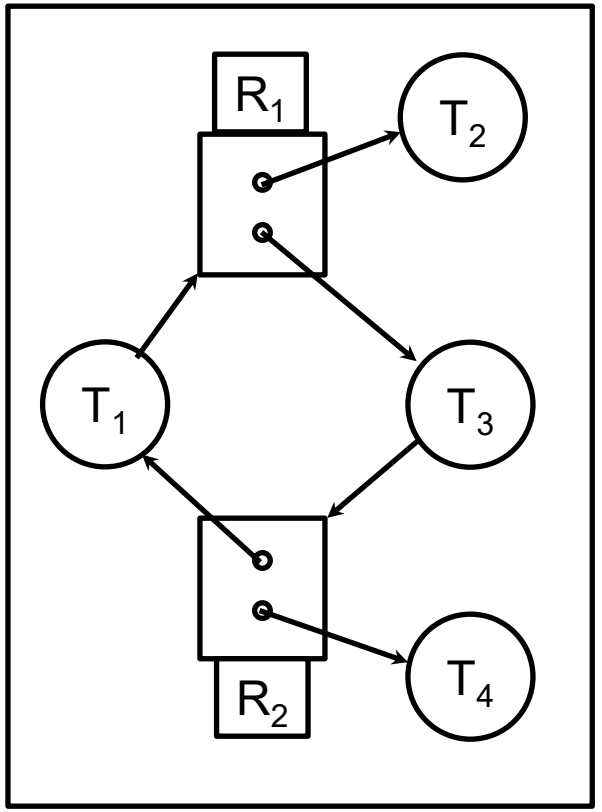
```
//Thread T2:
x = x - 2;
```

```
//Thread T3:
x = x / 2;
```

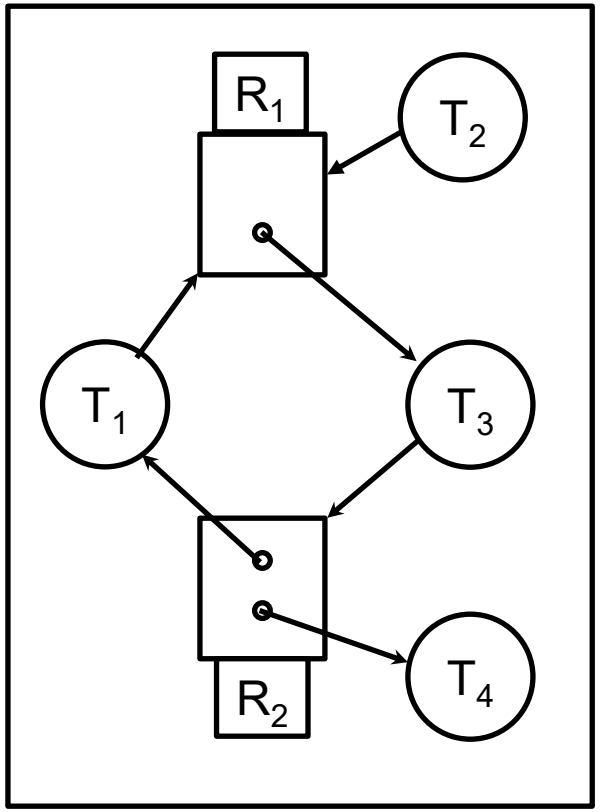
Q4 Deadlocks (20 pts)

- Consider the following Resource Allocation Graph with 4 processes and 2 resource types. Number of small circles in the box of resource Rj indicates the number of instances of resource Rj. An arrow from process Ti to resource Rj indicates that Ti requests 1 instance of Rj; an arrow from resource Rj to process Ti indicates that Ti is holding 1 instance of Rj.
- Run Banker's algorithm to check if the current state is safe, by writing out the matrices Max, Allocation and Need, and vectors Total and Available. If yes, give a safe sequence of process completions and fill in the table with the sequence of process completions without deadlock, and available resources after the completion of each process.

Q4 Deadlocks Con't (20 pts)



(a)



(b)

Max

	R1	R2
T1		
T2		
T3		
T4		

Allocation

	R1	R2
T1		
T2		
T3		
T4		

Need

	R1	R2
T1		
T2		
T3		
T4		

Total

R1	R2

Available

R1	R2

Available

	R1	R2
Init	0	0

Q4 Deadlocks (a) ANS

Max

	R1	R2
T1	1	1
T2	1	0
T3	1	1
T4	0	1

Allocation

	R1	R2
T1	0	1
T2	1	0
T3	1	0
T4	0	1

Need

	R1	R2
T1	1	0
T2	0	0
T3	0	1
T4	0	0

Total

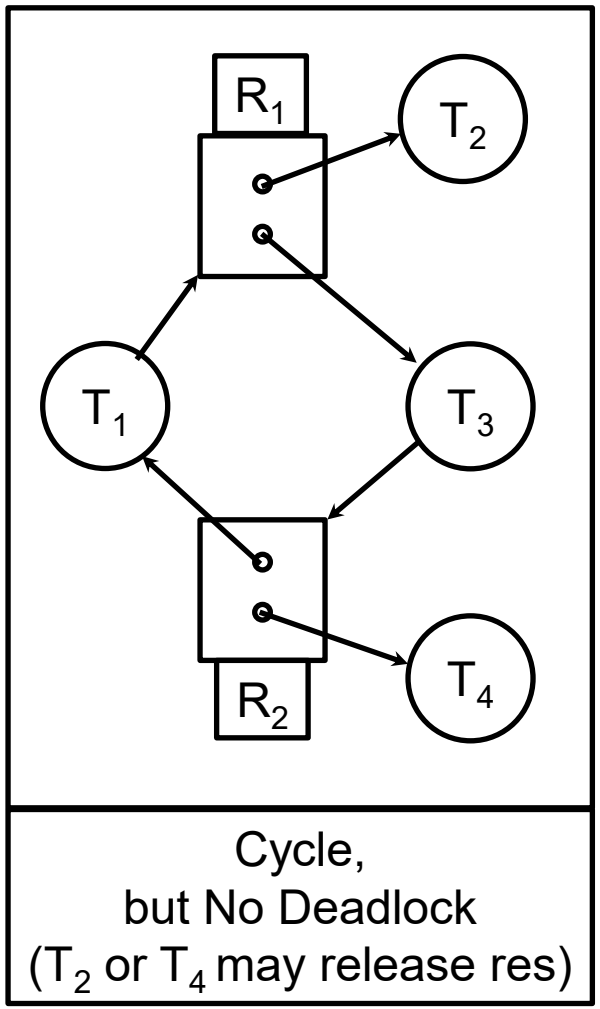
R1	R2
2	2

Available

R1	R2
0	0

Available

	R1	R2
Init	0	0
T4	0	1
T3	1	1
T1	1	2
T2	2	2



Cycle,
but No Deadlock
(T_2 or T_4 may release res)

No deadlock

Safe Sequence: T4, T3, T1, T2)
(or T2, T4, T1, T3)
Or other as T2, T4 can finish anytime

Q4 Deadlocks (b) ANS

Max

	R1	R2
T1	1	1
T2	1	0
T3	1	1
T4	0	1

Allocation

	R1	R2
T1	0	1
T2	0	0
T3	1	0
T4	0	1

Need

	R1	R2
T1	1	0
T2	1	0
T3	0	1
T4	0	0

Total

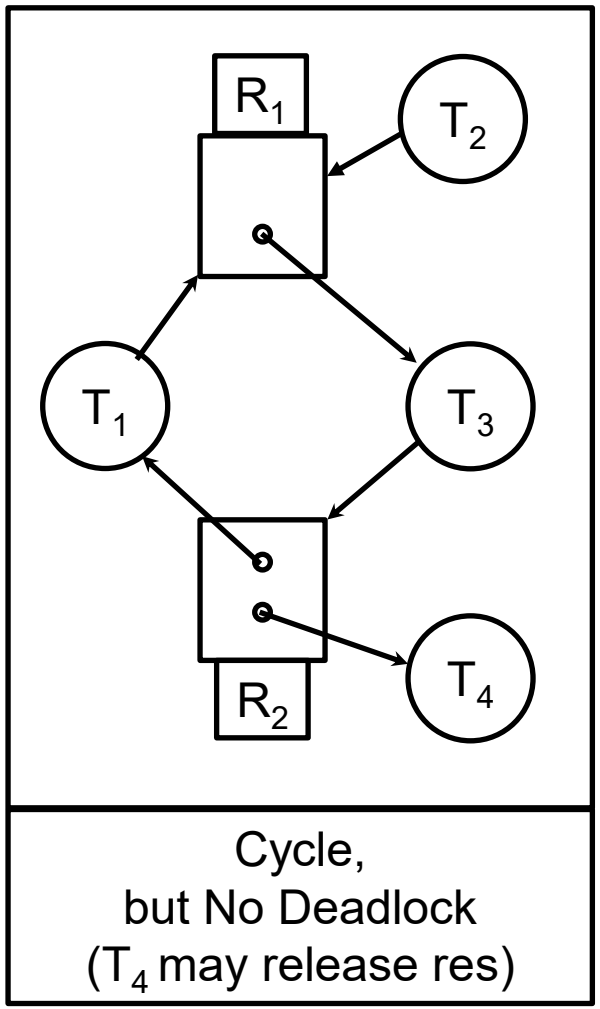
R1	R2
1	2

Available

R1	R2
0	0

Available

	R1	R2
Init	0	0
T4	0	1
T3	1	1
T1	1	2
T2	1	2



Cycle,
but No Deadlock
(T_4 may release res)

No deadlock

Safe Sequence: T4, T3, T1, T2)
(or T4, T3, T2, T1)

Q4 Deadlocks (c)

- Consider the Dining Lawyers problem. There are 3 lawyers P1 to P3, each with a different number of arms. P1 has 1 arm and needs 1 fork to eat; P2 has 2 arms and needs 2 forks to eat; P3 has 3 arms and needs 3 forks to eat. There is a pile of 3 forks at center of the table. Each lawyer picks up one fork at a time, and when he gets enough forks, he eats and then puts down all his forks. Use Banker's algorithm to check if it is possible for the system to be deadlocked. If no, give a safe sequence of process completions without deadlock. If yes, show an unsafe state that will result in a deadlock. (You need to give the Max, Allocation, Need matrices, Total and Available vectors, and Available resources after completion of each process.)

Q4 Deadlocks (c) ANS

Initially, all forks are free.

Max	Allocation
1	0
2	0
3	0

Total	Available
3	3

P2 grabs 1 fork and P3 grabs 2 forks

Max	Allocation	Need
1	0	1
2	1	1
3	2	1

Total	Available
3	0

Current state is a deadlock.

With 3 forks total, P2 holding 1 and P3 holding 2 leaves 0 available and each process still needs more forks

Available resources after completion of each process

	R1
Init	0
Deadlock	

Q4 Deadlocks (c) ANS (Thought Experiment)

Initially, all forks are free.

Max	Allocation
1	0
2	0
3	0

Total	Available
3	3

P1 grabs 1 fork and P2 grabs 2 forks

Max	Allocation	Need
1	1	0
2	2	0
3	0	3

Total	Available
3	0

Available resources after completion of each process

	R1
Init	0
P1	1
P2	3
P3	3

Current state is not a deadlock.

With 3 forks total, P1 holding 1 fork can finish, P2 holding 2 forks can finish, and P3 can then finish

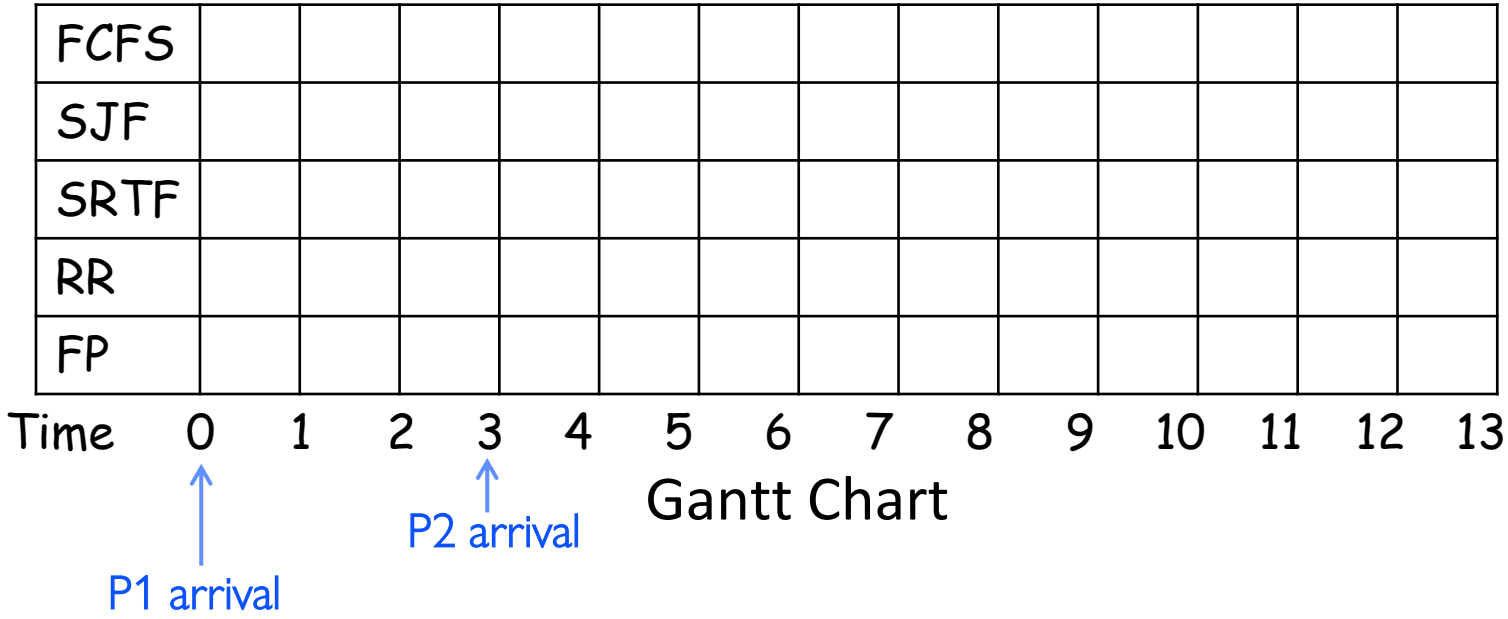
If you give this scenario in the exam, it is incorrect, because I asked for a deadlock scenario if one exists.

Q5 Scheduling (20 pts)

- Consider the set of 2 processes whose arrival time and CPU/IO burst times are given below. For each scheduling algorithm (FCFS, SJF, SRTF, RR, Fixed-Priority (FP)), draw the Gantt chart by filling in the table with the PID that runs in each time slot, and calculate the response time for each process, and the average response time. Assume that context switch overhead is 0. For RR scheduling, the time quantum is 1. For RR, assume that an arriving process is scheduled to run at the beginning of its arrival time, i.e., it is added to the head ofthe queue upon arrival. **For FP scheduling, assign P2 (PID 2) higher priority than P1 (PID 1). In case of a tie, prefer the running process to the newly arrived process, and if everything else is equal, prefer the process with smaller index.** If a time slot is idle with no active process executing, then fill in X. (Except for any possible idle time slots at the end of schedule, leave them empty and do not fill in X.)

Q5 Scheduling Cont'

PID	Arriv. time	CPU Burst	IO Burst	CPU Burst	FCFS Resp. Time	SJF Resp. Time	SRTF Resp. Time	RR Resp. Time	FP Resp. Time
1	0	5	3	2					
2	3	1	2	3					
					Avg RT	Avg RT	Avg RT	Avg RT	Avg RT



Q5 Scheduling ANS

PID	Arriv. time	CPU Burst	IO Burst	CPU Burst	FCFS Resp. Time	SJF Resp. Time	SRTF Resp. Time	RR Resp. Time	FP Resp. Time
1	0	5	3	2	10	10	11	11	11
2	3	1	2	3	10	10	6	6	6
					Avg RT 10	Avg RT 10	Avg RT 8.5	Avg RT 8.5	Avg RT 8.5

FCFS	1	1	1	1	1	2	x	x	1	1	2	2	2
SJF	1	1	1	1	1	2	x	x	1	1	2	2	2
SRTF	1	1	1	2	1	1	2	2	2	1	1		
RR	1	1	1	2	1	1	2	2	2	1	1		
FP	1	1	1	2	1	1	2	2	2	1	1		

Time 0 1 2 3 4 5 6 7 8 9 10 11 12 13

P1 arrival

P2 arrival

Gantt Chart

FCFS: Time 8: Tie broken to prefer P1 with smaller index
SJF: Time 8: P1 runs before P2 since its CPU burst of 2 is shorter than P2's CPU burst of 3

SRTF, RR, FP: at time 3, P2 preempts P1; after time 4, P1 and P2 have perfect overlap of CPU and IO bursts, so there is no CPU contention, and the 3 scheduling algorithms give the same trace