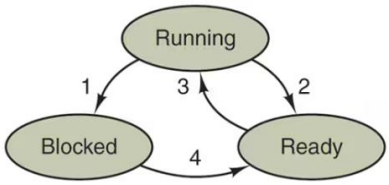
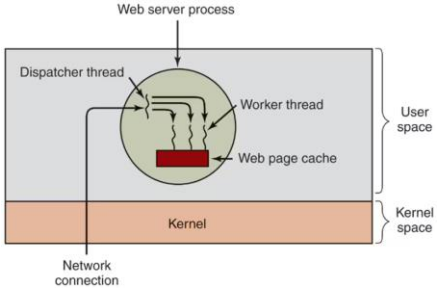
Assignment 2
CSC 4420 Computer Operating System
Chapter 2
Due: Feb. 16, 2025
50 Points

1. **We have discussed 3 process states in the class shown below. In theory, with three states, there could be six transitions, two out of each state. However, only four transitions are shown. Are there any circumstances in which either or both of the missing transitions might occur? (5 Points)**



- **Blocked -> Running:**
  - **When a blocked process that is waiting for I/O event to complete finishes its condition, it does not immediately execute**
  - **Instead moves to the Ready state, waiting for the scheduler to assign it in the CPU**
  - **The OS could allow an unblocked process to immediately run if only has a higher priority than the current process that is running but only in priority based systems.**
  - **So a process must first go through the ready state before running.**
- **Ready -> Blocked:**
  - **A process in Ready state cannot initiate an I/O request or block itself because does not have been given CPU time.**
  - **Only a process that is running currently can make blocking system call**
  - **So a process cannot block without running first.**

2. **The following figure shows a multithreaded Web server. If the only way to read from a file is the normal blocking read system call, do you think user-level threads or kernel-level threads are being used for the Web server? Why? (5 Points)**



- **User Level thread:**
  - **User level thread cannot handle blocking system calls, because the normal blocking read() will block the entire process not just the calling thread.**
- **Kernel level thread:**
  - **In the Kernel level thread when making a blocking system call, only that thread will be paused while other threads continue their execution.**

- **Kernel level thread allow efficient handling of blocking system calls because it ensures other threads to remain active**

3. **In the classical thread model, the register set is listed as a per-thread rather than a per-process item shown below. Why? After all, the machine has only one set of registers. (5 Points)**

Per process items	Per thread items
Address space Global variables Open files Child processes Pending alarms Signals and signal handlers Accounting information	Program counter Registers Stack State

- **One reason can be that OS must save and restore registers per thread not per process, because switching between threads is important for multitasking (meaning the correct states need to be restored). If all registers were per-process then all threads in a process would interfene with each other's state.**
- **And because each thread has its own execution context, registers are set as per-thread.**

4. **In this problem, you are to compare reading a file using a single-threaded file server and a multithreaded server. It takes 15 msec to get a request for work, dispatch it, and do the rest of the necessary processing, assuming that the data needed are in the block cache. If a disk operation is needed, as is the case one-third of the time, an additional 75 msec is required, during which time the thread sleeps. How many requests/sec can the server handle if it is single threaded? If it is multithreaded? (5 Points)**

Multithreaded servrer (wasted time of sleep can be used to handle other requests)

a. **Every 15ms a new request is handled by an available thread**
b. **Even when some threads sleep due to disk I/O new threads can start processing**
c. **The request per second -> (1000 ms / second)/ (15 ms / request) = 66.67 request/sec**

Single Threaded server (time is wasted during disk I/O

d. **If the data is in cache = 15ms**
e. **If the data is in disk operation = 15ms + 75ms = 90ms**
f. **The Average time per processing request -> T_avg = (2/3 * 15) + (1/3 * 90) = 30/3 + 90/3 = 120/3 = 40ms**
g. **The request per second -> (1000 ms / second)/ (40 ms / request) = 25 request/sec**

5. **Does Peterson's solution to the mutual-exclusion problem shown below work when process scheduling is preemptive? How about when it is nonpreemptive? (5 Points)**

```
#define FALSE  0
#define TRUE   1
#define N      2                        /* number of processes */

int turn;                               /* whose turn is it? */
int interested[N];                      /* all values initially 0 (FALSE) */

void enter_region(int process);         /* process is 0 or 1 */
{
     int other;                         /* number of the other process */

     other = 1 – process;               /* the opposite of process */
     interested[process] = TRUE;        /* show that you are interested */
     turn = process;                    /* set flag */
     while (turn == process && interested[other] == TRUE)  /* null statement */ ;
}

void leave_region(int process)          /* process: who is leaving */
{
     interested[process] = FALSE;       /* indicate departure from critical region */
}
```

- **Nonpreemptive**
    - **Nonpreemtive means that scheduling is cooperative, the CPU will be assigned a new process for execution.**
    - **Mutual-exclusion**
        - **If one process is in critical section, no other process is allowed to be executed in their critical section**
    - **Therefore Peterson's solution works because one process will loop until the other's process's turn is switched to FALSE.**
- **Preemptive**
    - **Preemptive means a running process can be interrupted at any time and replaced by another process / or current process can run again.**
    - **Because Peterson's Solution ensures mutual exclusion using flag and turn, which will prevent both processes entering critical section at the same time, it will make sure that turn and flags are always checked before entering critical section.**
    - **And then when preempted process resumes, it will continue from where it left off and recheck conditions.**
- **Therefore Peterson's solution works for both preemptive and non-preemptive scheduling.**

6. **The producer-consumer problem can be extended to a system with multiple producers and consumers that write (or read) to (from) one shared buffer. Assume that each producer and consumer runs in its own thread. Will the solution presented in the class as below, using semaphores, work for this system? (5 Points)**

```
#define N 100

typedef int sema;
sema mutex=1;
sema empty=N, full=0;

void producer(void){
 int item;
 while(TRUE){
  item = produce_item();
  down(&empty);
  down(&mutex);
  insert_item(item);
  up(&mutex);
  up(&full);
 }
}
```

```
void consumer(void){
                  int item;
                  while(TRUE){
   down(&full);
   down(&mutex);
   item = remove_item();
   up(&mutex);
   up(&empty);
   consume_item(item);
          }
}
```

- **The semaphore-based solution will work for multiple producers and consumers in terms of overflow and underflow -> mutual exclusion and buffer integrity**
- **But it may suffer from inefficiency if too many producers and consumers are active simultaneously, due to high contention on the mutex semaphore**
- **Mayne using separate mutexes for different operations could optimize the usage of multiple threads.**

7. Five batch jobs, *A* through *E*, arrive at almost the same time. They have estimated running times of 10,6,2,4, and 8 minutes. Their (externally determined) priorities are 3, 5, 2, 1, and 4, respectively, with 5 being the highest priority. For each of the following scheduling algorithms, determine the mean process turnaround time. Ignore process switching overhead.

a. **Round robin.**
b. **Priority scheduling.**
c. **First-come, first-served (run in order 10, 6, 2, 4, 8).**
d. **Shortest job first.**

For (a), assume that the system is multiprogrammed, and that each job gets its fair share of the CPU. For (b) through (d), assume that only one job at a time runs, until it finishes. All jobs are completely CPU bound. (10 Points)

a. Let Time Quantum = 2 with the following jobs: A(10), B(6), C(2), D(4), E(8)

Time	Job Executing	Remaining Times	Description
0	**A**	**A: 10, B: 6, C: 2, D: 4, E: 8**	**A runs for 2 minutes (remaining time = 8)**
2	**B**	**A: 8, B: 4, C: 2, D: 4 E: 8**	**B runs for 2 minutes (remaining time = 4)**
4	**C**	**A: 8, B: 4, C: 2, D: 4, E: 8**	**C runs for 2 minutes (remaining time = 0)**
6	**D**	**A: 8, B: 4, D: 4, E: 8**	**D runs for 2 minutes (remaining time = 2)**
8	**E**	**A: 8, B: 4, D: 2, E: 8**	**E runs for 2 minutes (remaining time = 6)**
10	**A**	**A: 8, B: 4, D: 2, E: 6**	**A runs for 2 minutes (remaining time = 6)**
12	**B**	**A: 6, B: 4, D: 2, E: 6**	**B runs for 2 minutes (remaining time = 2)**
14	**D**	A: 6, B: 2, D: 2, E: 6	**D runs for 2 minutes (remaining time = 0)**
16	**E**	**A: 6, B: 2, E: 6**	**E runs for 2 minutes (remaining time = 4)**
18	**A**	**A: 6, B: 2, E: 4**	A runs for 2 minutes (remaining time = 4)
20	**B**	**A: 4, B: 2, E: 4**	B runs for 2 minutes (remaining time = 0)
22	**E**	**A: 4, E: 4**	E runs for 2 minutes (remaining time = 2)
24	**A**	**A: 4, E: 2**	A runs for 2 minutes (remaining time = 0)
26	**E**	**A: 2**	E runs for 2 minutes(remaining time = 0)

Mean Turnaround Time = (24+20+4+14+26)/5 = 17.6 Minutes

b. Priority Scheduling

A = 10 minutes, Priority = 3
B = 6 minutes, Priority = 5
C = 2 minutes, Priority = 2
D = 4 minutes, Priority = 1
E = 8 minutes, Priority = 4

(Job with highest priority runs first)
B (Priority 5, 6 minutes) -> starts at 0, finish at 6 minutes
E (Priority 4, 8 minutes) -> starts at 6, finish at 14 minutes
A (Priority 3, 10 minutes) -> starts at 14, finish at 24 minutes
C (Priority 2, 2 minutes) -> starts at 24, finish at 26 minutes
D (Priority 1, 4 minutes) -> starts at 26, finish at 30 minutes

(Mean Turnaround Time)
Mean Turnaround Time = (6+14+24+26+30)/5 = 100/5 = 20 minutes

c. First-Come, First Serverd (FCFS) scheduling

A = 10 minutes -> starts at 0, finish at 10 minutes
B = 6 minutes -> starts at 10, finish at 16 minutes
C = 2 minutes -> starts at 16, finish at 18 minutes
D = 4 minutes -> starts at 18, finish at 22 minutes
E = 8 minutes -> starts at 22, finish at 30 minutes

Mean Turnaround Time = (10+16+18+22+30)/5 = 19.2 minutes

d. Shortest Job First Scheduling

(Smallest burst time is executed first)
A = 10 minutes -> **C** = 2 minutes -> start at 0, finish at 2 minutes
B = 6 minutes -> **D** = 4 minutes -> start at 2, finish at 6 minutes
C = 2 minutes -> **B** = 6 minutes -> start at 6, finish at 12 minutes
D = 4 minutes -> **E** = 8 minutes -> start at 12, finish at 20 minutes
E = 8 minutes -> **A** = 10 minutes -> start at 20, finish at 30 minutes

Mean Turnaround Time = (2+6+12+20+30)/5 = 14 minutes

8. A soft real-time system has four periodic events with periods of 50,100,200, and 250 msec each. Suppose that the four events require 35,20,10, and x msec of CPU time, respectively. What is the largest value of x for which the system is schedulable? (5 Points)

Total CPU utilization = U
U = sum{n}{i=1} (C_i)/(T_i)

C_i -> CPU time required by task i
T_i -> period of task i
n -> number of tasks

T_1 = 50 msec, C_1 = 35 msec
T_2 = 100 msec, C_2 = 20 msec
T_3 = 200 msec, C_3 = 10 msec
T_4 = 250 msec, C_4 = x msec

(If the total utilization is 1 or less, it means that the CPU is fully utilized, but not overburdened, otherwise the tasks require more CPU time than is available, which leads the system be unschedulable.
U = sum{n}{i=1} (C_i)/(T_i)
U = (35/50)+(20/100)+(10/200)+(x/250)

U = (0.95) + (x/250)
(0.95) + (x/250) <= 1
x/250 <= 0.05
x <= 12.5 milliseconds is the largest value of x for which system is schedulable

9. The readers and writers problem can be formulated in several ways with regard to which category of processes can be started when. Carefully describe two different variations of the problem, each one favoring (or not favoring) some category of processes (e.g., readers or writers). For each variation, specify what happens when a reader or a writer becomes ready to access the database, and what happens when a process is finished. (5 Points)

Variation	Reader's Behavior	Writer's Behavior	Potential Issue
Reader-Priority	Readers can always read unless a writer Is actively writing. Multiple readers can read simultaneously.	Writers must wait until all readers finish.	Writers can experience starvation if new readers keep arriving
Writer-Priority	Readers must wait if a writer is waiting.	Writers get immediate access once the resource is free.	Readers can experience starvation if writers keep arriving