**CS 471: Operating System Concepts**

**Spring 2016**

**Solution**

**Examination I**

**Points: 125**

**March 4, 2016**

**Time: 4 hour slot (1201am – 1159pm)**

Turning in this exam under your name confirms your continued support for the honor code of Old Dominion University and further indicates that you have neither received nor given assistance in completing it.

**Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**CS Unix ID:** [**\_\_\_\_\_\_\_\_\_\_\_@cs.odu.edu**](mailto:___________@cs.odu.edu)

|  |  |  |
| --- | --- | --- |
| **Question #** | **Points** | |
| **Maximum** | **Obtained** |
| **1** | **25** |  |
| **2** | **25** |  |
| **3** | **25** |  |
| **4** | **25** |  |
| **5** | **25** |  |
| **Total** | **125** |  |

ANSWER ALL QUESTIONS. SHOW YOUR WORK.Question 1. [10 Points]

Given the following set of processes---with the specified length of the CPU burst, arrival time, and priority---compute **waiting time for P1-P4** with **preemptive SJF** scheduling. (Note: Show the **Gantt chart** and other **working details** in your worksheet.)

|  |  |  |  |
| --- | --- | --- | --- |
| Process | CPU Burst time | Arrival time | Priority |
| P1 | 30 | 10 | 3 |
| P2 | 20 | 17 | 1 |
| P3 | 15 | 21 | 4 |
| P4 | 10 | 27 | 2 |

Answer.

0-10 CPU Idle

10-17 P1 23 remaining for P1; P1 preempted by P2

17-21 P2 16 remaining for P2; P2 preempted by P3

21-27 P3 9 remaining for P3; P3 continues

27-36 P3 P3 finishes

36-46 P4 P4 finishes

46-62 P2 P2 finishes

62-85 P1 P1 finishes

P1 waiting time = 85-10-30=45

P2 waiting time = 62-17-20= 25

P3 waiting time = 36-21-15=0

P4 waiting time=46-27-10=9

Question 2. [10 Points]

Given the following set of processes---with the specified length of the CPU burst, arrival time, and priority---**compute turnaround time for P1-P4** with Round-Robin scheduling with a time quantum of 15. (Note: Show the **Gantt chart** and other **working details** in your worksheet.)

|  |  |  |  |
| --- | --- | --- | --- |
| Process | CPU Burst time | Arrival time | Priority |
| P1 | 30 | 10 | 3 |
| P2 | 20 | 17 | 1 |
| P3 | 15 | 21 | 4 |
| P4 | 10 | 27 | 2 |

0-10 Idle

10-25 P1 15 remaining for P1

25-40 P2 5 remaining for P2

40-55 P3 P3 finished

55-70 P1 P1 finished

70-80 P4 P4 finished

80-85 P2 P2 finished

Turnaround time for P1 = 70-10=60

Turnaround time for P2 = 85-17=68

Turnaround time for P3 = 55-21=34

Turnaround time for P4 = 80-27=53

Question 3. [5 Points]

Given the following equation for predicting the length of the next CPU burst, and the actual burst lengths for the first 4 CPU bursts, determine **the prediction of the CPU burst for the fifth CPU burst.**

**τn+1 = α\*tn  + (1-α)\*τn with α=0.8 and τ1** = 3, **t1=5, t2=10, t3=15, t4=5(actual CPU bursts)**

Ans:

**τ2** = 0.8\*5 + 0.2\*3 = 4.6

**τ3** = 0.8\*10 + 0.2\*4.6 = 8.92

**τ4** = 0.8\*15 + 0.2\*8.92 = 13.784

τ5 = 0.8\*5 + 0.2\*13.784 = 6.7568

**Question 4. [Points 10]**

Given the following solution for the critical section problem (the given code is for process Pi; code for Pj is the same with i and j interchanged), show one scenario where it does not work correctly. Here, **flag[i] and flag[j] are shared** variables initialized to **false**. And **turn** is initialized to **i**.

do{

/\* Begin Entry Section \*/

flag[i]=true; turn=i;

while (flag[j] && (turn==j));  
 /\* End Entry section \*/  
   
 Critical section  
   
 /\* Begin Exit section \*/  
 flag[i]=false; turn=j;  
 /\*End exit section \*/  
   
 Remainder section  
 }

Ans:

T0: Pi: flag[i]=true; turn=i; so while fails;

T1: P0 enters CS;

T2: Pj: flag[j]=true; turn=j; so while fails;

T3: P1 enters CS;

So both Pi and Pj are in their CS at the same time violating the mutual exclusion condition.

**Question 5. [10 Points]**

Referring to Fig. 5.18, monitor solution code for the dining-philosophers problem, with the five philosophers numbered 0-4, EXPLAIN whether or not philosophers 2 and 3 can eat simultaneously? Assume that all other philosophers are in THINKING state. Justify referring to specific parts of the relevant code.

Ans:

No. Philosophers 2 and 3 cannot eat simultaneously. Philosophers 0,1, and 4 are in THINKING state.

At this time 2 and 3 are also in THINKING state.

Let us say Philosopher 2 first executes pickup(2). Then state[2]=HUNGRY.

test(2) is called. Since state[1] and state [3] are not EATING, so state[2]=EATING.

Now Philosopher 3 executes pick(3). Then state[3]=HUNGRY.

test(3) is called. Since state[3]==EATING, the test fails.

In summary, in this scenario, when Philosopher 2 was successful in going to EATING state, the test method prevented Philosopher 3 also to simultaneously go in to EATING state.

Question 6. [5 Points]

Following is a code for swapping the values in two variables. Two processes P0 and P1 are manipulating the integer variables X and Y in a shared memory.  Initially, X=20 and Y = 5. Assuming that P0 and P1 execute simultaneously (i.e., not necessarily sequentially) on a single CPU, show two possible values of X and Y after both P0 and P1 complete. For each value, show how the details of the scenario where it could occur (in the worksheet).

**Void Swap(int \*a, int \*b) { int temp = \*a; \*a = \*b; \*b = temp;}**

Case 1: Suppose P0 executes first and then P1:

Initially: X=20, Y=5

Swap(X,Y)

temp=20

X=5

Y=20

Now P1 executes:

Swap(X,Y)

Temp=5

X=20

Y=5

Final value: X=20, Y=5

Case 2:

X=20, Y=5

P0: Swap(X,Y): temp=20; a=5; P0 preempted------X=5, Y=5

P1: Swap(X,Y); temp =5; a=5; b=5; X=5, Y=5

P0: Resumes: b=temp=20;; X=5, Y=20

Final value, X=5, Y=20.

Several other possibilities exist.

Question 7 [Points 10]

Given the following information about resources and processes, construct a resource-allocation graph and a wait-for graph. Also, determine whether or not there is a deadlock. Justify your answer.

|  |  |  |  |
| --- | --- | --- | --- |
| Resource | #of instances | Requested by | Allocated to |
| R1 | 2 | P1, P2 | P3, P4 |
| R2 | 1 | P1, P4 | P2 |
| R3 | 1 | P4 | P3 |
| R4 | 2 | P2 | P1 |

Resource allocation graph: Here, Since R4 has two resources and one of them is allocated to P1, the other one could be allocated to R4. The following allocation graph reflects this allocation also.





Wait-for graph:

P1 is waiting for R1 and R2: So it is waiting on P2, P3, P4

P2 is waiting for R1; So it is waiting on P3, P4

P3 is not waiting for any resource;

P4 is waiting for R2; so it is waiting on P3.

No deadlock: There is no cycle; Also, P3 can finish first; releasing R1 and R3; P2 can get R1 and finish and release R1, R2 and R4. P1 can get R1 and R2 and finish and release R1, R2, R4. Now P4 can finish.

**Question 8 [Points 10]**

In the following scenario, suppose we have avoided circular-wait by ordering the resources in the order R1 < R2 < R3 < R4 (R1 first, R2 next, and so on) and enforcing that each process must request resources in the **ascending** order. Show one **possible allocation** of R1-R4 to P1-P4 and the corresponding wait-for-graph. Is there a deadlock? Justify.

|  |  |  |  |
| --- | --- | --- | --- |
| Resource | #of instances | Requested by | Allocated to |
| R1 | 2 | P1, P2 |  |
| R2 | 1 | P1, P3 |  |
| R3 | 1 | P3, P4 |  |
| R4 | 2 | P2, P4 |  |

Ans:

P1 requests: R1 < R2

P2 requests: R1 < R4

P3 requests: R2 < R3

P4 requests: R3 < R4

One possible scenario:

R1 allocated to P1, P2

R2 allocated to P3; so P1 has to wait

R3 allocated to P4;

R4 allocated to P2, P4;

|  |  |  |  |
| --- | --- | --- | --- |
| Resource | #of instances | Requested by | Allocated to |
| R1 | 2 | P1, P2 | P1, P2 |
| R2 | 1 | P1, P3 | P3 |
| R3 | 1 | P3, P4 | P4 |
| R4 | 2 | P2, P4 | P2, P4 |



P1 is waiting for R2 on P3;

P2 not waiting;

P3 waiting for R3 on P4

P4 not waiting;

NO deadlock as there is no cycle; P3,P4 finish; Then P2 can finish; then P1 can finish

**Question 9 [5 Points]**

Consider a single resource (R1) system with 6 instances of R1. Currently, there are 3 processes in the system. Their maximum need and the current allocations are specified below. Also, specified are new requests for more of R1 from these three processes. Show an order in which the processes should be granted the resources so the system is always in a safe state.(Hint: Use Banker's algorithm)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Process ID | Maximum need | Current allocation | New request | **Need** |
| P1 | 6 | 2 | 1 | **6-2=4** |
| P2 | 4 | 1 | 1 | **4-1=3** |
| P3 | 3 | 1 | 1 | **3-1=2** |

Ans:

Available=2

Since P3 can complete with the current available resources, grant P3’s request. Eventually it will complete execution and release its resources.

Available=2+1=3.

Since P2 can complete with the current available resources, grant P2’s request. Eventually it will complete execution and release its resources.

Available=3+1=4

Since P1 can complete with the current available resources, grant P1’s request. Eventually it will complete execution and release its resources.

**So the requests should be granted in the order: P3 first; after it completes; P2 next; after P2 completes; P1 last**

**Question 10 [8 Points]**

Given the following information about the current state of main memory, determine the **memory location**s to which two processes P1 and P2 requiring 6.5K and 4K memory, respectively, should be allocated based on (i) **Best-fit strategy and (ii) Worst-fit strategy**. You must first allocate memory to P1 and then to P2. Show the status of the memory after the allocation (for P2) in each case. Current memory status: **0-6K allocated; 6K-12K: Free; 12K-24K allocated; 24K-35K: Free; 35K-40K allocated; 40K-55K Free.**

**Ans:**

**Best-fit:**

P1: Needs 6.5K; Available memory fragments: 6K, 11K, 15K;

So allocate 24K-31K to P1; this results in a new free fragment 30.5K-35K;

P2: Needs: 4K; Available: 6K, 4.5K, 15K

So allocate 30.5K-35K

**Worst-fit:**

P1: Needs 6.5K; Available memory fragments: 6K, 11K, 15K;

So allocate 40K-47K to P1; this results in a new free fragment 46.5K-55K;

P2: Needs: 4K; Available: 6K, 11K, 8.5K

So allocate 24K-28K

**Question 11 [Points 10]**

Suppose a memory manager employs paging with page size of 1024 bytes (=1K bytes). It has a memory of 16 Mbytes. A process of size 6.5 Kbytes needs to be loaded into memory. Answer the following. (a) How many frames are there in the memory? (b) How many bits are necessary to represent the physical address as <frame#, offset>? (c) How many frames need to be allocated to the process?

Ans: (a) 16\*220/210 = 16\*210  = 214  =16K frames

(b) For 16K frames, we need 14 bits for frame#. For 1K byte frame, we need 10 bits to represent offset. A total of 24bits.

(c) 7 Frames

**Question 12 [7 Points]**

In a memory management system that employs both TLB (access time 70 nanoseconds) and main memory (120 nanoseconds), with a TLB hit ratio of 0.75, determine the effective memory access time.

Ans: 0.75\*(70+120) nano + 0.25\*(70+120+120)=220 nanosec

**Question 13 [Points 10]**

Determine the **number of page faults** for the following sequence of 15 page references by a process assuming demand paging and **LRU page replacement policy**. The process has been allocated 4 frames. Page reference string: 5,6,7,5,1,7,6,8,1,7,1,5,6,1,5

Pages **5,6,7** result in page fault : Pages 5,6,7 are in memory

5 no page-fault Pages 5,6,7 are in memory

**1 results** page fault Pages 1,5,6,7 are in memory

7,6 no page-fault Pages 1,5,6,7 are in memory

**8 result**s in page fault Pages 1,6,7,8 are in memory

1,7,1 no page fault Pages 1,6,7,8 are in memory

**5** **results** in page fault Pages 1,5,7,8 are in memory

**6 results** in page fault Pages 1,5,6,7 are in memory

1,5 no page fault Pages 1,5,6,7 are in memory

Number of page faults= 7

**Question 14 [10 Points]**

Determine the **number of page faults** for the following sequence of 15 page references by a process assuming demand paging and **Optimal page replacement policy**. The process has been allocated **3 frames**. Page reference string: 5,6,7,5,1,7,6,8,1,7,1,5,8,1,5

Ans:

Pages **5,6,7** result in page fault : Pages 5,6,7 are in memory

5 no page-fault Pages 5,6,7 are in memory

**1 results in** page fault Pages 1,6,7 are in memory

7,6 no page-fault Pages 1,6,7 are in memory

**8 result**s in page fault Pages 1,7,8 are in memory

1,7,1 no page fault Pages 1,7,8 are in memory

**5** **results** in page fault Pages 1,5,8 are in memory

8 nopage fault Pages 1,5,8 are in memory

1,5 no page fault Pages 1,5,6,7 are in memory

Number of page faults= 6

**Question 15 [5 Points]**

With demand paging (in virtual memory), if main memory access time is 100 nanoseconds and time to service a page fault is 0.1 milliseconds, determine effective access time (in milliseconds) in a situation where 2 of a set of 20 references result in page-fault. [Hint 1 millisecond = 106 nanoseconds]

Effective access time = 18/20\*100 (nano) + 2/20\*(0.1 millisec) = 90+10000=10090 nanosec or 0.01009 milliseconds.