**Operating Systems**

What is an Operating system

Client –server and peer-to-peer computing

Dual mode operation, timer

Operating system structure

**Processes**

Process states

Process creations

Interprocess communication

Reasons

Shared-memory system

Message passing system

Problem 1. The following program contains no syntax errors. As it executes it will create one or more processes.

Simulate the execution of this program and **show clearly how processes are created**

#include<stdio.h>

main()

{

int x = 5, y = 2, z = 30;

x = fork();

y = fork();

if(x ! = 0) printf(“Type 1\n”);

if(y ! = 0) printf(“Type 2\n”);

z = fork();

if((x>0) || (y>0) || z>0)) printf(“Type 3\n”);

if((x==0) && (y==0) && (z != 0)) printf(“Type 4\n”);

if((x!=0) && (y!=0) && (z != 0)) printf(“Type 5\n”);

if((y!=0) && (z == 0)) printf(“Type 6\n”);

}

p2==0) && (p3 == 0)) printf(“type 5\n”);

}

Also answer the following questions

How many processes are created?

\_\_\_\_\_\_\_\_\_8\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

How many times will this program print the following?

“Type 1” \_\_\_2\_\_\_\_\_\_\_\_\_\_\_\_\_

“Type 2” \_\_\_\_2\_\_\_\_\_\_\_\_\_\_

“Type 3” \_\_\_\_\_7\_\_\_\_\_\_\_\_\_\_\_

“Type 4” \_\_\_\_\_\_1\_\_\_\_\_\_\_\_\_\_

“Type 5” \_\_\_\_\_\_\_1\_\_\_\_\_\_\_\_

“Type 6” \_\_\_\_\_\_\_\_2\_\_\_\_\_\_\_

**Multithreading**

Benefits of multithreaded programming

Multithreading models

**Scheduling algorithms**

Scheduling Criteria

FCFS

SJF

SRTF

Priority

Problem 1. Given the following set of processes with, arrival times, burst times, priorities

|  |  |  |  |
| --- | --- | --- | --- |
| **Process** | **Arrival time** | **Burst time** | **Priority** |
| **P1** | **0** | **9** | **4** |
| **P2** | **1** | **6** | **2** |
| **P3** | **2** | **1** | **1** |
| **P4** | **3** | **5** | **3** |
| **P5** | **5** | **2** | **2** |

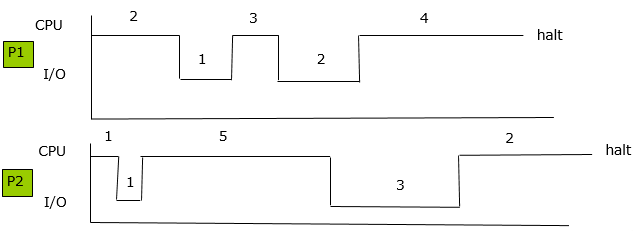
1. Draw four Gant charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF (Shortest-Job First)(non-preemptive), non-preemptive priority, preemptive Shortest-Remaining–time first
2. What is the turnaround time of each process for each of the scheduling algorithms in part 1?
3. What is the waiting time of each process for each of the scheduling algorithms in part 1?
4. What is the average waiting time over all processes for each of the scheduling algorithms in part 1?

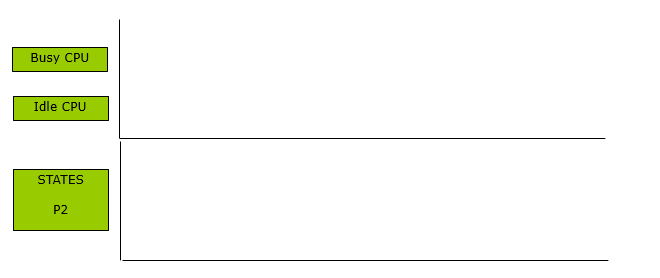
**Round Robin**

**The CPU and I/O times for 2 processes are shown below. Assume that PI gets to the ready queue just before P2 and the scheduling algorithm used by the OS is Round Robin with a time slice of 3 time units. Assume that the I/Os for the processes are different so that there is no I/O queue. Assume also that an interrupt from a completed I/O for process "X" will place process "X" in the ready queue BEHIND the process that was just interrupted.**

Using the first empty graph, describe how the CPU will be assigned to each process and for how long.

Use the second empty graph to show ALL the states that P1 goes through and the amount of time it has remained in that state until it has halted.

****

****

**Deadlocks**

Necessary Conditions for deadlock

Methods for handling deadlocks

Prevention

Avoidance

Resource allocation graph algorithm

Banker’s algorithm

Detection

Wait-for-graph

Deadlock-detection algorithm

Problem 1. Draw the wait-for graph for the following situation:

P1 is using R1 and waiting for R2

P2 is using R2 and waiting for R4

P3 is using R5 and R3 and is waiting for R1

P4 is using R4 and is waiting for R5

1. Is there deadlock in this system?
2. If the answer is YES , then which processes are deadlocked

Problem 2. A system has: 5 processes *P*0 through *P*4; and 3 resource types: *A* (9 instances), *B* (5 instances), and *C* (7 instances). At a time T0 the state of the system is as shown below

***Allocation Max Available Need***

***A B C A B C A B C A B C***

***P*0 1 1 0 6 5 3 2 3 1 5 4 3**

***P*1 1 0 2 3 2 2 2 2 0**

***P*2 2 0 1 80 2 6 0 1**

***P*3 2 1 1 3 2 2 1 1 1**

***P*4 1 0 2 4 2 3 3 2 1**

1. Is the system is in a safe state?

**P1 P3 P4 P0 P2**

**P3 P1 P4 P0 P2**

**P3 P1 P4 P2 P0**

**………………..**

1. Can request for (2,2,1) by *P*4 be granted? If YES find the safe state. If NO explain why NOT

***Allocation Max Available Need***

***A B C A B C A B C***

***P*0 1 1 0 6 5 3 0 1 0**

***P*1 1 0 2 3 2 2**

***P*2 2 0 1 80 2**

***P*3 2 1 1 3 2 2**

***P*4 3 2 3 4 2 3**

1. Can request for (0,2,0) by *P*0 be granted? If YES find the safe state. If NO explain why NOT

***Allocation Max Available Need***

***A B C A B C A B C***

***P*0 1 3 0 6 5 3 2 1 1**

***P*1 1 0 2 3 2 2**

***P*2 2 0 1 80 2**

***P*3 2 1 1 3 2 2**

***P*4 1 0 2 4 2 3**

1. Can request for (1,0,1) by *P*2 be granted? If YES find the safe. If NO explain why NOT

***Allocation Max Available Need***

***A B C A B C A B C A B C***

***P*0 1 1 0 6 5 3 1 3 0**

***P*1 1 0 2 3 2 2**

***P*2 3 0 2 80 2**

***P*3 2 1 1 3 2 2**

***P*4 1 0 2 4 2 3**

**Process Synchronization**

Requirements of solution to the critical-section

Peterson’s solution

The bounded-buffer problem

Problem 1. A producer and a consumer threads are sharing a buffer as shown below. The buffer is initialized with “blank” characters in each position. Semaphores X, Y, and Z are used to protect the buffer and are initializing properly. After the threads have been running for a while, the buffer appears as shown below and the threads are executing at the location shown by the arrows below. If the last character placed in the buffer was ‘8’ and the value of semaphore Y is 5, answer the question below.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **2** | **a** | **8** | **d** | **f** | **1** | **9** | **s** | **m** | **c** | **w** | **7** | **b** |

Consumer

Repeat

do other calculation

P(X)

P(Z)

Read item from buffer

V(Z)

Do other calculation

V(Y)

Until false

Producer

Repeat

do other calculation

P(Y)

P(Z)

Place item in buffer

V(Z)

Do other calculation

V(X)

Until false

What was the value of semaphore X when it was initialized? \_\_\_\_\_\_\_0\_\_\_\_\_\_\_\_\_

What is the value of semaphore X now? \_\_\_\_\_\_\_\_\_\_7\_\_\_\_\_\_\_

What is the value of the consumer buffer index now? \_\_\_\_\_\_\_\_9\_\_\_\_\_\_\_\_\_

 What is the value of the producer buffer index now? \_\_\_\_\_\_\_\_\_3\_\_\_\_\_\_\_\_

Which was the last character read by the consumer? \_\_\_\_\_\_\_\_\_m\_\_\_\_\_\_\_\_

Which specific character that are still in the buffer HAVE BEEN READ by the consumer

\_\_\_\_\_\_\_\_\_\_\_df19sm\_\_\_\_\_\_\_\_\_\_\_\_\_

**Note. Initialization: X (full)= 0, Y(empty)=13, Z(mutex) = 1**

**Problem 2.** Suppose that processes P1, P2, and P3 shown below running concurrently. S1 and S2 are among the statements that P1 will eventually execute, S3 and S4 are among the statements that P2 will eventually execute and S5 and S6 are among the statements that P3 will eventually execute.

You need to use semaphores to guarantee that statement S3 is executed AFTER statement S6 has been executed – also that statement S1 will be executed BEFORE statement S4 – also that statement S5 is executed BEFORE statement S2.

Show, within the structure of the processes below, how you would use semaphores to coordinate these three processes. Include semaphore names and to which values you would initialize the semaphores.

1. Which semaphores are you using and their initial values: Sem1=1, Sem2=1, Sem3=1

Process P1 Process P2 Process P3

S1 S3 S5

S2 S4 S6

1. After the semaphores have been appropriately placed, is it possible to determine which of the 6 statements above will be executed first?

If YES – which one? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. And which of the 6 statements above will be executed last?

If YES – which one \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Quiz solution**

Q1. (1 point)

Describe the differences among short term, medium term, and long term scheduling.

Answer:

**Long-term scheduler** (or job scheduler) – selects which processes should be brought into the ready queue

**Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU

**Medium-term scheduler** – removes processes from memory to reduce the degree of multiprogramming.

Q2. (2 points)

Describe three different models of establishing relationship between user-level and kernel-level threads.

Answer:

Three common ways of establishing a relationship between user threads and kernel threads are:

1. Many-to-One
2. One-to-One
3. Many-to-Many

Q3. (1 point)

Describe possible states in which a process can be. Draw the state diagram corresponding to these states.

Answer: As a process executes, it changes *state*

* **new**: The process is being created
* **running**: Instructions are being executed
* **waiting**: The process is waiting for some event to occur
* **ready**: The process is waiting to be assigned to a processor
* **terminated**: The process has finished execution

Q4. (1 point)

Name activities are performed for OS during context switching.

Answer:

1. Answer: Save state of P1
2. Service interrupt
3. Select next user process P2
4. Restore state of P2 and restart CPU

Q5. (2 points)

1. Compare the overhead of context switching between two threads in different processes and two processes. Explain

Answer:

Context-switching between two threads is faster than between two processes.

Context-switching between two threads in different processes will be the same as switching between two processes.

1. What are the advantages and disadvantages of user-level and kernel-level threads?

Answer:

Kernel-level:

Adv: Kernel sees every thread of every process. If a thread makes a system call, other threads within that process can run. Takes advantage of multiprocessors

Disadv: Switching among threads of a process is done via interrupts

Context switching is slower than for user-level threads but faster than process context switching.

User-level:

Adv: Switching among threads of a process is done by calls to library modules and is done very

quickly

Disadv: If a thread makes a system call, other threads within that process will be blocked.

Does not take advantage of multiprocessors

Q6. (1 point)

In a system with threads, is there normally one stack per thread or one stack per process? Explain.

Answer:

One stack per thread