**Operating Systems**

**Quiz Scheduling – Spring 2016**

1. Consider the following set of processes with arrival times and CPU burst times.

|  |  |  |
| --- | --- | --- |
| **Process** | **Arrival time** | **Burst time** |
| P1 | 0 | 7 |
| P2 | 3 | 6 |
| P3 | 4 | 2 |
| P4 | 8 | 8 |

Draw a Gantt chart for the CPU schedule for each of the following process scheduling algorithms, and calculate the wait time for each process for each algorithm. Show your work.

* Round-robin (quantum of 3)
* Shortest job first pre-emptive
* Shortest job first non-pre-emptive

1. Examine the following correct program. Assume that the OS is using a FCFS scheduling algorithm, with the following details:

* Actions that can move a thread from running state to wait state are any forced waits (like on a semaphore or join) and a request for I/O.
* As soon as a thread is created, it is put on the ready queue. When post is called on a semaphore, a single one of any threads waiting on the semaphore is chosen at random and put on the ready queue. When a thread exits, all threads waiting on that action with a join are moved onto the ready queue.
* Explain what order the marked lines (lines R, S, T, W, X, Y, Z) will get executed in and why. Show the state of the ready queue at each line.

int i, x = 1;

sem\_t A, B, C;

void \*thread\_A( void \*arg ) {

sem\_wait( &A );

x = 2; // **LINE X**

sem\_post( &A );

pthread\_exit( NULL );

}

void \*thread\_B( void \*arg ) {

sem\_wait( &B );

x = 3; // **LINE Y**

sem\_post( &B );

pthread\_exit( NULL );

}

void \*thread\_C( void \*arg ) {

sem\_wait( &C );

x = 4; // **LINE Z**

sem\_post( &C );

pthread\_exit( NULL );

}

int main() {

pthread\_t tid[3];

sem\_init( &A, 0, 1 );

sem\_init( &B, 0, 1 );

sem\_init( &C, 0, 1 );

sem\_wait( &C );

sem\_wait( &B );

sem\_wait( &A );

pthread\_create( &tid[0], NULL, thread\_C, NULL ); **// LINE R**

pthread\_create( &tid[1], NULL, thread\_A, NULL ); **// LINE S**

pthread\_create( &tid[2], NULL, thread\_B, NULL ); **// LINE T**

sem\_post( &B );

sem\_post( &C );

sem\_post( &A );

x = 5; **// LINE W**

for (i = 0; i < 3; i++ ) pthread\_join( &tid[i] );

}

1. Examine the following correct program. Below it are 4 different versions of the code for **threadA**. Compare and contrast how the program will behave in each of the cases, by explaining precisely how the program will behave in each case and why.

sem\_t A, B;

int main()

{

pthread\_t threads[THREADS];

int i;

sem\_init( &A, 0, THREADS );

sem\_init( &B, 0, 1 );

for ( i = 0; i < THREADS; i++ )

pthread\_create( &threads[i], NULL, threadA, NULL );

for ( i = 0; i < THREADS; i++ )

pthread\_join( threads[i], NULL );

printf( "Finished\n" );

pthread\_exit( 0 );

}

**VERSION 1**

void \*threadA( void \*ign )

{

sem\_wait( &A );

printf( "1\n" );

sleep( 3 );

sem\_wait( &B );

printf( "2\n" );

sleep( 3 );

sem\_post( &B );

sem\_post( &A );

pthread\_exit( NULL );

}

**VERSION 2**

void \*threadA( void \*ign )

{

sem\_wait( &A );

printf( "1\n" );

sleep( 3 );

sem\_post( &A );

sem\_wait( &B );

printf( "2\n" );

sleep( 3 );

sem\_post( &B );

pthread\_exit( NULL );

}

**VERSION 3**

void \*threadA( void \*ign )

{

sem\_wait( &B );

printf( "1\n" );

sleep( 3 );

sem\_post( &B );

sem\_wait( &A );

printf( "2\n" );

sleep( 3 );

sem\_post( &A );

pthread\_exit( NULL );

}

**VERSION 4**

void \*threadA( void \*ign )

{

sem\_wait( &B );

printf( "1\n" );

sleep( 3 );

sem\_wait( &A );

printf( "2\n" );

sleep( 3 );

sem\_post( &A );

sem\_post( &B );

pthread\_exit( NULL );

}

1. The shortest job first (SJF) scheduling algorithm is optimal with respect to process waiting time. Explain what makes it difficult to use in reality, and is there any way to partially overcome that issue?
2. In question 1, you imagined a simplification of the SJF pre-emptive scheduling algorithm, with no execution quantum. In reality, pre-emptive scheduling algorithms have an execution quantum associated with them. Please explain why it is needed.
3. Pre-emptive scheduling algorithms in general suffer from a problem. Please name and explain that problem, and name and explain the technique used to overcome it.
4. Consider the following process arrival, CPU, and I/O burst times given. Assume that there is a single I/O device which operates in FCFS manner.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Process** | **Arrival time** | **CPU Burst 1** | **I/O Burst 1** | **CPU Burst 2** | **I/O Burst 2** |
| P1 | 0 | 5 | 8 | 6 | 8 |
| P2 | 5 | 5 | 10 | 3 | 9 |
| P3 | 6 | 5 | 12 | 3 | 10 |

Draw the Gantt charts for the CPU schedule and the I/O device schedule for the FCFS CPU scheduling algorithm.