CSC 4420

Computer Science Operating Systems

Due 6/24/2024 – 11:59pm

Assignment 2

8 questions - 25 points

Answer the following 8 questions. Give an exhaustive motivation with all the necessary details.

Question 1 (2 points): The register set is listed as a per-thread rather than a per-process item in the following table. Why? After all, the machine has only one set of registers.

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

Question 2 (2 points): What is the meaning of the term busy waiting? What other kinds of waiting are there in an operating system? Can busy waiting be avoided altogether? Explain your answer.

Question 3 (3 points): The program shown below uses the Pthreads API. What would be the output from the program at LINE C and LINE P? Motivate your answer.

```
#include <pthread.h>
#include <stdio.h>
#include <types.h>

int value = 0;
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
pid t pid;
pthread t tid;
pthread attr t attr;
        pid = fork();

        if (pid == 0) { /* child process */
                pthread attr init(&attr);
                pthread create(&tid,&attr,runner,NULL);
                pthread join(tid,NULL);
                printf("CHILD: value = %d",value); /* LINE C */
        }
        else if (pid > 0) { /* parent process */
                wait(NULL);
                printf("PARENT: value = %d",value); /* LINE P */
        }
}

void *runner(void *param) {
        value = 5;
        pthread exit(0);
}
```

Question 4 (3 points): Consider the following solution to the mutual-exclusion problem involving two processes P0 and P1. Assume that the variable turn is initialized to 0. Process P0's code is presented below.

```
/* Other code */

while (turn != 0) { }; /* Entry Section. */

Critical Section;
```

```
turn = 0;  /* Exit Section*/

/* Other code */
```

For process P1, replace all 0 by 1 in above code. Determine if the solution meets all the three required conditions for a correct critical section problem solution.

Question 5 (3 points) Does the busy waiting solution using the turn variable (see figure below) work when the two processes are running on a shared-memory multiprocessor, that is, two CPUs sharing a common memory? Is there anything else necessary compared to the single-CPU case?

```
while (TRUE) {
    while (turn != 0)        /* loop */ ;
    critical_region( );
    turn = 1;
    noncritical_region( );
}
```

(a) Process P0

```
while (TRUE) {
    while (turn != 1)        /* loop */ ;
    critical_region( );
    turn = 0;
    noncritical_region( );
}
```

(b) Process P1

Question 6 (3 points): Show that, if the wait() and signal() semaphore operations are not executed atomically, then mutual exclusion may be violated.

Question 7 (3 points): Race conditions are possible in many computer systems. Consider a banking system that maintains an account balance with two functions: deposit(amount) and withdraw(amount). These two functions are passed the amount that is to be deposited or withdrawn from the bank account balance. Assume that a husband and wife share a bank account. Concurrently, the husband calls the withdraw() function and the wife calls deposit(). Describe how a race condition is possible and what might be done to prevent the race condition from occurring. NOTE: assume

a “current_balance” variable is shared to store the current account balance. You should write the source code of your solution.

Question 8 (3 sub-questions 2 points each, 6 points total):

Suppose we have an atomic hardware instruction **TestAndSet** whose action is described by the pseudocode:

```
boolean TestAndSet (boolean *target) {
    boolean rv = *target;
    *target = TRUE;
    return rv;
}
```

The following pseudocode claims to implement mutual exclusion using TestAndSet:

```
SHARED:
   boolean lock = ???;

| line | Process 0:                                | line | Process 1:                                |
| 0.0  |                                           | 1.0  |                                           |
| 0.1  |    while (TRUE) {                         | 1.1  |    while (TRUE) {                         |
| 0.2  |       while (TestAndSet(&lock)) ;         | 1.2  |       while (TestAndSet(&lock)) ;         |
| 0.3  |       /* CRITICAL SECTION */              | 1.3  |       /* CRITICAL SECTION */              |
| 0.4  |       lock = FALSE;                       | 1.4  |       lock = FALSE;                       |
| 0.5  |       /* REMAINDER SECTION */             | 1.5  |       /* REMAINDER SECTION */             |
| 0.6  |    }                                      | 1.6  |    }                                      |
```

a. **(2 points)** Assume the lock is initialize to False. Suppose the scheduler happens to cause statements to be executed in the order shown below. Complete the blank columns in the table.

line	lock value	Explain non-obvious occurrences
0.0	???	Execution begins with Process P0
0.1		
0.2		
0.3		
0.4		
0.5		
0.6		P0 is interrupted; P1 is placed in RUNNING state
1.0		
1.1		
1.2		
1.3		
1.4		
1.5		
1.6		End of tracing

b. **(2 points)** Assume the lock is initialize to False. Suppose the scheduler happens to cause statements to be executed in the order shown below. Complete the blank columns in the table.

line	lock value	Explain non-obvious occurrences
0.0	???	Execution begins with Process P0
0.1		
0.2		
0.3		P0 is interrupted; P1 is placed in RUNNING state
1.0		
1.1		
1.2		
1.3		
. . .		End of tracing

c. **(2 points)** Assume the lock is initialized to True. What happens to the two processes?