Zagazig university
Faculty of Engineering
Computer and Systems Dept.

Spring 2018
3rd Year
CSE 325: Operating Systems

Assignment #1

Covers: Chapters 1 – 5

Due Date: Thursday 22/3/2018

Questions:

Solve the following questions:

Question 1:

Describe the differences between symmetric and asymmetric multiprocessing. What are three advantages and one disadvantage of multiprocessor systems?

Question 2:

What is the purpose of interrupts? How does an interrupt differ from a trap? Can traps be generated intentionally by a user program? If so, for what purpose?

Question 3:

Why is the separation of mechanism and policy desirable?

Question 4:

What are the five major activities of an operating system with regard to file management?

Question 5:

The Collatz conjecture concerns what happens when we take any positive integer n and apply the following algorithm:

$$n = \begin{cases} \frac{\mathrm{n}}{2}, \text{if n is even} \\ 3 \times \mathrm{n} + 1, \text{if n is odd} \end{cases}$$

The conjecture states that when this algorithm is continually applied, all positive integers will eventually reach 1. For example, if n = 35, the sequence is

35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1

Write a C program using the fork() system call that generates this sequence in the child process. The starting number will be provided from the command line. For example, if 8 is passed as a parameter on the command line, the child process will output 8, 4, 2, 1. Because the parent and child processes have their own copies of the data, it will be necessary for the child to output the sequence. Have the parent invoke the wait() call to wait for the child process to complete before exiting the program. Perform necessary error checking to ensure that a positive integer is passed on the command line.

Question 6:

In Question 5, the child process must output the sequence of numbers generated from the algorithm specified by the Collatz conjecture because the parent and child have their own copies of the data. Another approach to designing this program is to establish a shared-memory object between the parent and child processes. This technique allows the child to write the contents of the sequence to the shared-memory object. The parent can then output the sequence when the child completes. Because the memory is shared, any changes the child makes will be reflected in the parent process as well.

This program will be structured using POSIX shared memory as described in Section 3.5.1. The parent process will progress through the following steps:

a) Establish the shared-memory object (shm open(), ftruncate(), and mmap()).
b) Create the child process and wait for it to terminate.
c) Output the contents of shared memory.
d) Remove the shared-memory object.

One area of concern with cooperating processes involves synchronization issues. In this exercise, the parent and child processes must be coordinated so that the parent does not output the sequence until the child finishes execution. These two processes will be synchronized using the wait() system call: the parent process will invoke wait(), which will suspend it until the child process exits.

Question 7:

Using Amdahl's Law, calculate the speedup gain of an application that has:

a) 10 percent parallel component and two processing cores
b) 10 percent parallel component and eight processing cores
c) 90 percent parallel component and two processing cores
d) 90 percent parallel component and eight processing cores

Question 8:

A system with two dual-core processors has four processors available for scheduling. A CPU-intensive application is running on this system. All input is performed at program start-up, when a single file must be opened. Similarly, all output is performed just before the program terminates, when the program results must be written to a single file. Between startup and termination, the program is entirely CPU-bound. Your task is to improve the performance of this application by multithreading it. The application runs on a system that uses the one-to-one threading model (each user thread maps to a kernel thread).

a) How many threads will you create to perform the input and output? Explain.
b) How many threads will you create for the CPU-intensive portion of the application? Explain.

Question 9:

The program shown below uses the Pthreads API. What would be the output from the program at LINE C and LINE P?

```
#include <pthread.h>
#include <stdio.h>
#include <types.h>
int value = 10;
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 */
    {
        printf("CHILD: value = %d",value); /* LINE C */
    }
    else if (pid > 0)   /* parent process */
    {
        pthread attr init(&attr);
        pthread create(&tid,&attr,runner,NULL);
        pthread join(tid,NULL);
        wait(NULL);
        printf("PARENT: value = %d",value); /* LINE P */
    }
}
void *runner(void *param)
{
    value = 5;
    pthread exit(0);
}
```

Question 10:

The first known correct software solution to the critical-section problem for *n* processes with a lower bound on waiting of *n - 1* turns was presented by Eisenberg andMcGuire. The processes share the following variables:

```
enum pstate {idle, want in, in cs};
pstate flag[n];
int turn;
```

All the elements of flag are initially idle. The initial value of turn is immaterial (between 0 and n-1). The structure of process Pi is shown in the code below. Prove that the algorithm satisfies all three requirements for the critical-section problem.

```
Do {
    while (true) {
        flag[i] = want in;
        j = turn;
        while (j != i) {
            if (flag[j] != idle) {
                j = turn;
                else
                    j = (j + 1) % n;
            }
            flag[i] = in cs;
            j = 0;
            while ( (j < n) && (j == i || flag[j] != in cs))
                j++;
            if ( (j >= n) && (turn == i || flag[turn] == idle))
                break;
        }
        /* critical section */
        j = (turn + 1) % n;
        while (flag[j] == idle)
            j = (j + 1) % n;
        turn = j;
        flag[i] = idle;
        /* remainder section */
    } while (true);
```

To Do

You have to submit your answers for the previous questions in one file. For the questions that require coding, you need to provide the code as well as screen-shots of the output.

Submission

On 22/3/2018, the writeup of your solution should be submitted. Follow the following instructions carefully, otherwise your submission will not be accepted:

- Covert your solution to a pdf file and name it "CSE325_SP18__Assign1_YOURNAME.pdf", where YOURNAME is your name.
- Attach the pdf file into an email message which have the subject "CSE325 SP18 Assignment1" exactly.
- In the body of the message include your full name and bench number. You could also add any notes or special instructions.
- Send the message to the email address of your teaching assistant s.khalil9191@gmail.com

Important Notes:

This assignment must be done individually; any act of plagiarism will be penalized and reported.