EEE335 – Assignment 2

Note:

- **Questions are from chapter 2 of the course textbook (Tanenbaum, 4th Ed)**
- **For questions 1 through 5, show your work/calculations, as applicable.**
- **Assignments are to be completed and submitted individually.**

Chapter 2

1. [2 pts] Round-robin schedulers normally maintain a list of all runnable processes, with each process occurring exactly once in the list. What would happen if a process occurred twice in the list? Can you think of any reason for allowing this?

2. [8 pts total] Five batch jobs A through E, arrive at a computer center at almost the same time. They have estimated running times of 10, 6, 2, 4, and 8 minutes. Their (externally determined) priorities are 3, 5, 2, 1, and 4, respectively, with 5 being the highest priority. For each of the following scheduling algorithms, determine the mean process turnaround time. Ignore process switching overhead.

   a) [2 pts] Round robin;
   b) [2 pts] Priority scheduling;
   c) [2 pts] First-come, first-served (run in order A,B,C,D and E); and
   d) [2 pts] Shortest job first.

   For (a), assume that the system is multi-programmed, and that each job gets its fair share of the CPU. For (b) through (d), assume that only one job at a time runs, until it finishes. All jobs are completely CPU bound.

3. [4 pts] A process running on Compatible Time Sharing System (CTSS) needs 30 quanta to complete. How many times must it be swapped in, including the very first time (before it has run at all)? In your calculation, count the first time the process is assigned to the CPU.

4. [2 pts] Explain how time quantum value and context switching time affect each other, in a round-robin scheduling algorithm.

5. [4 pts] Consider a real-time system with two voice calls of periodicity 5 msec each with CPU time per call of 1 msec, and one video stream of periodicity 33 ms with CPU time per call of 11 msec. Is this system schedulable? Show your work.

Programming exercises

For programming exercise, you will need a Linux computer. It is recommended you to install Linux Ubuntu in VirtualBox. Once Linux is installed, you will need to install the MPI libraries. To do this, enter the following commands in your terminal:

```
$ sudo apt-get update
$ sudo apt-get install openmpi-bin openmpi-doc libopenmpi-dev
```

6. [10 pts] The value of π is approximately 3.14159265358979323846264 3. This value can be calculated using a sequence as follows:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define NUM_STEPS 1000

int main(int argc, char *argv[]) {

    int num_steps;
    if (argc == 1)
        num_steps = NUM_STEPS;
    else
        num_steps = atoi(argv[1]);

    const double PI25DT = 3.14159265358979323846264 3;
    double pi = 0;
    double error;
    double step = 1.0 / (double)num_steps;

    for (int i = 0; i < NUM_STEPS; i++) {
        double x = step * ((double)i + 0.5);
        pi = pi + 4.0 / (1.0 + x*x);
    }
    pi = pi * step;

    printf ("The calculated value of pi is: %20.16f, Error is %20.16f\n",
               pi, fabs(pi-PI25DT));
    printf ("Number of steps used: %d\n", NUM_STEPS);
}
```

Design an MPI program that calculates π using several processes to speed up calculation. Configure your virtual machine to use multiple cores and calculate the run time for different process numbers and number of increments (num_steps). To compile your MPI program and run it using 4 processes use the following commands:

```
$ mpicc -o mpi_pi mpi_pi.c -lm
$ mpirun -n 4 ./mpi_pi
```

7. [10 pts] In this MPI program, the processes print a message on the screen before the synchronization barrier and another after. Run the program using the commands indicated in comment at the beginning of the code. You will notice that the messages "**BEFORE**" and "**AFTER**" are not out of order. This is guaranteed by the call `MPI_Barrier()`. Observe what happens if you remove this command.

The `MPI_Barrier()` command is a global synchronization mechanism. At low level, it can be implemented using several point-to-point communications. In the following program, replace the `MPI_Barrier()` command with `MPI_Send()` and `MPI_Recv()` calls. Make sure the messages "**BEFORE**" and "**AFTER**" are not out of order.

```
/*
* To compile and run this program use the following commands
* mpicc -o mpi_barrier mpi_barrier.c
* mpirun -n 10 ./mpi_barrier
*/
#include <stdio.h>
#include <mpi.h>

int main(int argc, char *argv[])
{
    int myid, numprocs;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);

    for (int i=0; i<5; i++)
        printf("I am process %2d of %2d - BEFORE\n", myid, numprocs);

    MPI_Barrier(MPI_COMM_WORLD);

    for (int i=0; i<5; i++)
        printf("I am process %2d of %2d - AFTER\n", myid, numprocs);

    MPI_Finalize();
    return 0;
}
```

8. [10 pts] Design an MPI program that averages a series of 1,000,000 random numbers between 0 and 100. In your program, process 0 must initialize the series before sending it to other processes. The numbers will be kept in an array of doubles to ensure accurate calculations. At the end, process 0 should display the result on the screen.

9. [Bonus 5 pts] Design an MPI program that works exactly as in the previous question, but calculates the standard deviation instead of the average. Calculate the standard deviation σ of a population of n elements as follows where μ is the mean:

$$\sigma = \sqrt{\frac{\sum_{i=0}^{n-1}(x_i - \mu)^2}{n}}$$

Submission

For this assignment you will need to submit the following:

1. A single PDF document that includes your answers to the questions (include both the questions and answers) and your source code;
2. A separate .c file of your source code

The above files need to be submitted via a single compressed file (e.g. .zip or .7zip) with a meaningful name (e.g. `eee335assignment2_Your_last_name.zip`) and submitted via the course Moodle site. Recall that you set up the share folder in your home directory which allows you to share files between the host and guest OS.