CS 3305A: Operating Systems
Department of Computer Science
Western University
Assignment 2
Fall 2018
Due Date: November 2, 2018

Purpose

The goals of this assignment are the following:

- Get experience with *pthread_create()*, and *pthread_join()* POSIX Threads functions
- Understand how a thread works and what problems arise from multithreaded computing
- Understand void pointers and how to cast them
- Gain more experience with the C programming language

Specification for Resource Management Program

In this assignment you are to implement a resource management simulator. The simulator will utilize multithread programing to execute multiple tasks simultaneously while keeping track of the system's memory resources. You will be provided with skeleton code that will initialize the system resources and provide a stack of jobs for you to run. You will be tasked with managing the system resources as well as running those jobs on different threads.

Your simulator must handle the following:

- **Allocating Memory**: every task that is spawned will require an *x* amount of memory. You are to allocate the amount of memory needed by reserving that amount for that task while it runs.
- **Deallocating Memory**: once a task is completed you are to free the memory allocated for that task so other tasks can be spawned.
- You will need to utilize the POSIX Threads functions to execute the tasks.
- You're not to use mutexes to solve any synchronization issues. Those issues will be tackled in the next assignment.

When running the jobs you must use the following helper functions to generate your results:

- `print_exceed_memory`: executed when the required memory for the job exceeds the maximum amount of memory available. If this case occurs, then discard the job.

- `print_insufficient_memory`: executed when the required memory for the job exceeds the amount of memory that is currently available. If this case occurs, then push the job back on the jobs stack.

- `print_starting`: executed once the job starts running.

- `print_completed`: executed once the job is completed.

Provided Files

- Your changes should only be inside the simulate.c, simulate.h, files and Makefile
- You are encouraged to create other files to help with modularization of your code
- There are tests provided with the code. You are encouraged to run them and create your own tests as well

Use the following commands to compile and run the program:

Compile the program using:

```
make
```

Run the program using:

```
./myOS simulator input.txt
```

Test the program using:

```
make test
```

Definitions and Function footprints

```
#define SYSTEM_OUTPUT "system.out"
```

system.out will be the output file that your program will output its result to.

Hint: create a file pointer and pass it to the print helper functions.

```
#define NUMBER_OF_THREADS 4
```

You will need to initialize multiple threads to run the jobs. Use this value to create an array of p_threads.

```
void simulate(int, linked_stack_t*);
```

This is the footprint of the simulate functions. It will receive an integer that will define the amount of memory that your system will manage. The stack will contain all the jobs that you will need to run.

Structures and Typedefs

```
typedef struct
{
      int number, required_memory, required_time;
} job_t;
```

job_t will hold the values of each job.

```
typedef struct
{
      struct linked_list *head, *tail;
      int size;
} linked_stack_t;
```

linked_stack_t defines the stack that will be used to hold the jobs.

```
void push(linked_stack_t*, void*);
void* pop(linked_stack_t*);
```

push and pop can be used to insert and remove values off of the stack. Notice how push receives a void pointer? The value must be casted into a job_t pointer for you to be able to access its values.

Sample Output

```
Starting job #2
Starting job #1
Starting job #4
Starting job #3
Job #1 completed
Job #2 completed
Job #3 completed
Job #4 completed
Starting job #5
Unable to run job #6. Memory requirements exceed system capabilities
Starting job #7
Job #5 completed
Job #7 completed
```

Assignment Submission Guideline

- Must run on GAUL. Otherwise, you will recieve a mark of zero. Refer to the website for further information
- Only .c, .h, and Makefiles. Marks will be deducted if other files are submitted
- DO NOT compress your submission

Resources

- https://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html