CSE344 – System Programming
Homework#4 – Report

Berk Pekgöz
171044041

Solving The Problem & Design:

To explain solving the problem and design, there are 5 subjects:

- Thread H
- Threads of student-for-hire
- Main Thread
- Error and Ctrl + C Handling

Thread H:

In thread h function, first the file given in command line argument is opened, and homework queue is initialized. Then starts reading file character by character in a loop until end of file reached or money is become zero. After reading each character, that character added to the queue as a homework.

The queue is a simple linked list queue which has only initialize, add, get(remove) and free functions.

If the character is an invalid priority character (not Q/S/C) then program gives an error and terminates with freeing resources.

To achieve synchronization, thread h uses hw_mx semaphore to reach queue and money(which is global variable). Also posts queueempty semaphore to achieve producer consumer type problem solving.

In the end thread h posts an end barrier semaphore to make program sure to thread h reached end.(Because thread h is detached thread, and we cannot join this thread)

Threads of student-for-hire:

This thread has very simple job. First gets its own information structure from global students array with given student number to thread parameter. Then starts waiting start_work semaphore to do homework. If it can pass through that semaphore and it is marked as available, it means program is terminating.

If that student marked as not available then that means that thread has to sleep for 6 – speed second.

To achieve synchronization, this threads uses both hw_mx and stdnt_mx semaphores. Hw_mx for reaching money and stdnt_mx for updating student status.

After sleeping (sleeping only used for achieving solving homework, not for synchronization), thread posts available_stdnt semaphore to inform main thread to there is another thread which is available to solve homework.

Main Thread:

Main thread first initializes students array with reading student information file and other initializes (like semaphore / signal masking). After that students threads and thread h are created. Thread h is detached with pthread_detach().

After the initializations, main thread starts a loop which is actual loop of running the program. In that loop first waits for queueempty semaphore to be sure queue is not empty(queue can be empty in only one situation which will be explained later). After that gets a homework from queue and then waits for available_stdnt semaphore to be sure there is at least one student who is available. After passing that point, calls get_suitable_student() function to get most suitable and available student for given homework priority.

Then, updates that student's information and status with posting start_work semaphore of that student. And loop continues with this order until queue is empty or H cannot afford the next homework.

And finally program returns all resources back (free/destroy semaphore/join threads) and terminates program.

To achieve synchronization, this threads uses all semaphores created. Hw_mx for reaching money and homework queue. stdnt_mx for updating student status and getting available student for next homework. Queueempty to be sure there is at least one homework in queue. Available_stdnt to be sure there is at least one available student. H_end_barrier semaphore for being sure thread h reached end when exiting.

Error and Ctrl + C Handling:

In case of error, that thread marks terminated(global variable) as TRUE(defined as 1).

In case of user interrupt, interrupt handler marks both terminated and user_int variables as TRUE. Important point is "only main thread can catch SIGINT because SIGINT is masked before creating other threads and then restored as unmasked in main thread. With this way we know that which thread will catch interrupt, so checking becomes easier.".

If terminated variable is TRUE, thread h stop executing loop and terminates. But student threads terminated with pthread_cancel() from main thread with terminate_program() function.

In the loop of main thread there are check points at beginning of loop and after all sem_wait() functions to check if there is an error or user interrupt. With this way error checks of sem_wait() functions are achieved. Because only way of sem_wait error is user interrupt in my program and I checked for it.

Notices:

- All requirements mentioned in PDF have been achieved.
- Input file must be in exactly same structure with given example file.
- Maximum length of line in input file is defined as 100 at the beginning of the code. If the lines are longer please change it.
- Maximum length of student name in input file is defined as 50 at the beginning of the code. If the names are longer please change it.
- All kinds of synchronization between processes achieved without mutexes and condition variables.
- Main thread clean up after all threads.
- In case of an arbitrary error, exit by printing to stderr a nicely formatted informative message and free resources.
- Some function calls does not have error check because they have no valid error in our processes.
- In case of CTRL-C the program (that means all threads) stop execution, return all resources to the system and exit with an information message.
- If the required command line arguments are missing/invalid, program print usage information and exit.
- No memory leak with valgrind.