Systems Software
Assignment 2: Creating Libraries, Makefile, and Dynamic Memory Allocation

Assignment Objectives

Students should demonstrate the following abilities:

1. Write C code that can be reused in the form of a library
2. Write makefile to build C programs with multiple source files
3. Use dynamic memory allocation C functions to create dynamic data structures

Assignment

The scheduler in one the important programs in an operating system. The scheduler is responsible for allocating the CPU to processes or programs loaded in memory. The scheduler selects one program among the set of programs eligible to run on the CPU, based on some criteria. One of the criteria used by the scheduler is to select the programs in the order of their arrival time (loading time or request to run) or in FiFO order (First In First Out). To implement such criterion, the scheduler uses a queue data structure to store the list of programs waiting for the CPU. The scheduler removes one program from the queue to run on the CPU and removes the next when the current program is completed.

You are asked to create a C program to simulate the FIFO scheduler program. Implement the following steps to complete the program.

1. Define a structure **`process`** that represents the information of a process. The structure contains three data members: **`process id`**, **`process arrival time`**, and **`process execution time`**. Define two functions to manipulate variables of type **`process`**: **`print_process`** to print the information of a process and **`compare_processes`** to compare two processes and return **`0`** if their ids are the identical, or **`1`** otherwise. Create header and implementation files for the structure **`process`** (`process.h` and `process.c`).
2. Define a structure **`queue_node`** that represents one node in the data structure **`queue`**. A queue is implemented as a doubly linked list (see slides on assignment 3 on courseSite). queue contains one data member of type **`process`** and two pointers to type **`queue_node`**, one named **`next`** and the other **`previous`**. Define five functions to manipulate elements in a queue: **`enqueue()`** to add a node at the back, **`dequeue()`** to remove a node from the front, **`search()`** to find a process in the queue, **`print_queue()`** to print the list of processes in the queue, and **`size()`** to

return the number of nodes in the queue. Create header and implementation files for the structure **`queue_node`** (`queue.h` and `queue.c`).

3. Write a main C program **`scheduler.c`** that is described in the steps below:

    a. The input data arrives at different times. To simulate the arrival times of the processes, write a function **`process_arrived`** that accepts a parameter **`time`** and returns a variable of type **`process`**. If the input parameter **`time`** matches one the arrival times in the table below, the method returns a **`process`** variable with the information of the process that arrived at **`time`**. The definition of the method is provided below. Use it as is.

Process id	Execution time (ms)	Arrival time (ms)
1	100	15
2	50	25
3	180	50
4	10	75
5	75	100
6	35	125
7	200	130

```
process process_arrived(int time){
  process p;
  p.id = 0;p.arrival_time=0; p.execution_time=0;
  switch(time){
    case 15:p.id = 1; p.arrival_time = 15; p.execution_time = 100;
            break;
    case 25:p.id = 2; p.arrival_time = 25; p.execution_time = 50;
            break;
    case 50:p.id = 3; p.arrival_time = 50; p.execution_time = 180;
            break;
    case 75:p.id = 4; p.arrival_time = 75; p.execution_time = 10;
            break;
    case 100:p.id = 5; p.arrival_time = 100; p.execution_time = 75;
             break;
    case 125:p.id = 6; p.arrival_time = 125; p.execution_time = 35;
             break;
    case 130:p.id = 7; p.arrival_time = 130; p.execution_time = 200;
             break;
  }
  return p;
}
```

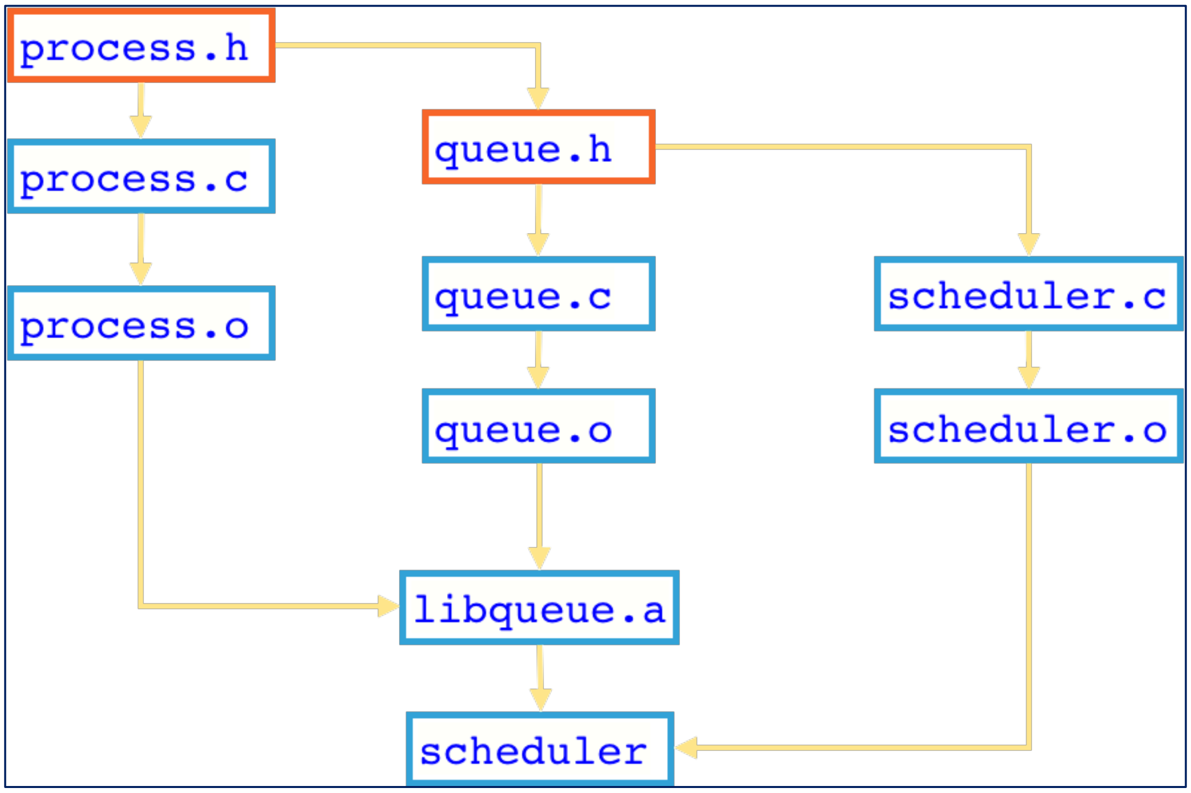
b. Write a simulation loop in the main function for time from 0 to 665 (total of the execution times). The code structure of the loop is shown below using comments.

```
for(int time = 0; time < SIMULATION_TIME; time++){
      // call process_arrived() to check if a process arrived
      // if a process arrived, enqueue it in the queue

      // if there is a current process running and the current
      // execution time is not elapsed, then decrement
      // the current execution time

      // if there is a current process running and its execution
      // time is elapsed, then print the process information,
      // the completion time (time), and the waiting time
      // (time – execution time – arrival time)
      // reset current process id to 0 (no process running)

      // if there is no current process running and
      // the queue is not empty, then dequeue a process and set
      // the current running process and the current execution
      // time to the dequeued process and execution time
 }
```

4. Your program should output statistics similar to the following.

```
Id          Time        Arrived     Completed         Waiting
1           100         15          115               0
2           50          25          165               90
3           180         50          345               115
4           10          75          355               270
5           75          100         430               255
6           35          125         465               305
7           200         130         665               335
```

Organize your code in four directories `/src`, `/include`, `/lib`, and `bin/`under the main directory **`/scheduler`**. Create a `makefile` to build your program executable as described by the dependency chart below. Create the static library `libqueue.a` in the folder `/lib` to combine **`process.o`** and **`queue.o`** and link it with the executable.



Submit the zipped folder **`scheduler.zip`** on courseSite.