GENERAL PROJECT STATEMENT

Write a C language program on linux, which simulates the environment in Unix where new processes are continually arriving, existing processes are vying for the CPU, processes are using their given quantum (CPU bound) or blocking because of I/O operations, and processes are terminating when their work is finished.

THE main() FUNCTION LOOP

The main() function will:

- Initialize the process table
- Loop for 99 processes, repeatedly doing the following (not necessarily in this order):
  - Processing a clock tick (interrupt); the clock is 60Hz so it ticks (interrupts the CPU) 60 times a second; the process table will be appropriately updated EACH clock tick; so each loop through the main() function is 1 clock tick
  - Terminating the ruNning process if it has reached its maximum CPU time (MAX Time)
  - Preempting the ruNning process if it blocked or used all of its quantum, recalculating its priority, and immediately placing it at the END of the proper process queue, based on its new priority
  - Scheduling the next process if necessary, scheduling ROUND ROBIN WITHIN EACH priority queue, where -1 is the highest priority, -2 the next highest, ..., -n, 0, 1, ..., n (i.e., n is the lowest priority)
  - Making Ready any Blocked processes that have become unblocked
  - Placing any newly arrived processes into the process table
- The loop stops when the process ID (PID) reaches the value 100
- THE ABOVE LOOP IS DONE EVERY CLOCK TICK

THE PROCESS TABLE

Information kept on each process (this matches the columns on the screen output):

- PID: Process ID, assigned from 1 to 99 as each new process arrives
- CPU Used: Total clock ticks used by the process so far
- MAX Time: Maximum CPU time needed by the process (in clock ticks); note, this is known for the process when it begins
- STATE: Process state; N, B, or R (ruNning, Blocked, or Ready)
- PRI: Process priority, 0=highest to 5=lowest, negative means Blocked; when a process becomes unblocked, it retains its negative priority until scheduled to the CPU, and thus will be scheduled first; priority -1 is scheduled before -2, etc.
- QUANTUM USED: Amount of the quantum used by the process (in clock ticks)
- BLK TIME: Blocking time; clock ticks used until the process blocks (1-6):
  - 1 to 5 means the process is I/O bound and blocks before using all of its quantum; 1 is highly I/O bound through 5 which is only a slightly I/O bound process
  - 6 indicates a CPU bound process (does no I/O); it always uses all of the quantum it is given and never blocks
- WAIT TKS: The total number of clock ticks the process has been waiting in the Ready state since it was placed into the process table

THE PRIORITY CALCULATION FORMULA

To recalculate the priority of the ruNning process, use the following formula:

new_priority (PRI) = (|old_priority| + quantum_used) / 2

ROUNDING to the nearest whole number; if the process blocked, set this priority negative, otherwise it will be positive

THE PROGRAM INITIALIZATION FUNCTION (what ALL new processes ALWAYS begin with)

PID: the next number in sequence (starting with 1)
CPU Used: 0 (zero); no CPU time used yet
MAX Time: a random integer between 1 and 18 (inclusive)
STATE: R (Ready to use the CPU)
PRI: 0 (zero)
QUANTUM USED: 0 (zero)
BLK TIME: a random integer between 1 and 6 (inclusive) where 6 means 100% CPU bound (uses all CPU time it is given); note, there is a 1 in 3 chance this will be set to 6, and a 2 in 3 chance this will be set to 1 through 5
WAIT TKS: 0 (zero); has not been waiting for the CPU

MISCELLANEOUS OTHER INFORMATION (use where necessary)

There can be a maximum of 10 processes in the process table; new processes will not arrive if the process table is currently full
There may be at most 1 processed marked as ruNning (state N); all other processes are either Ready (state R) or Blocked (state B)
There are 6 clock ticks in 1 quantum and 60 clock ticks per second (60Hz clock); so a quantum is 100 milliseconds
Each time a process is scheduled to the CPU, it may use at most 1 quantum
At the beginning of the main() function, an initialize function is called; it initializes the process table with exactly 5 starting (Ready) processes
The rest of the main() function (the loop) calls functions to accomplish the tasks it needs; this loop will terminate, and the scheduling will end, when PID 100 becomes the next process to enter the queues
At EACH clock tick (that is, EVERY time through the main() function loop):
   There is a 1 in 20 chance EACH Blocked process becomes unblocked
      (EVERY Blocked process is given this chance to become unblocked)
   There is a 1 in 5 chance a new process will be added
      Example in C of how this is done (note, int rand() is in <stdlib.h>):

```
if (rand()%5)
   /* true part - 1 to 4 returned (80% chance), no new process added */
else
   /* false part - 0 returned (20% chance), a new process is added   */
```

A process becomes Blocked when it has used the number of clock ticks that is equal to its blocking time (BLK TIME), and its blocking time never changes
A process is terminated (removed from the process table) when it reaches its MAX Time
All processes are kept in a single process table by priority (PRI); the process table is ALWAYS displayed BEFORE and AFTER scheduling the CPU, with priorities shown from highest to lowest (-1, -2, -3, ..., -n, 0, 1, 2, ..., n); all processes with the same priority are kept in the same queue
In addition to the normal program operation, your program will be STRESS tested against the following two conditions:
   1. All processes in the process table become Blocked; the output should be:
      BEFORE: The last unblocked process is ruNning and has used its BLK TIME
      AFTER: All processes are blocked (there are no Ready nor ruNning processes)
      BEFORE: One or more processes have become Ready (or are newly arrived)
      AFTER: The highest priority Ready process is now ruNning
   2. The process table becomes empty (Number of Processes = 0); the output should be:
      BEFORE: The last process is ruNning and reached its MAX Time (is terminating)
      AFTER: The process table is empty (only the 2 heading lines are printed)
      BEFORE: A new Ready process has arrived
      AFTER: The newly arrived process is now ruNning

Note, every BEFORE is followed by an AFTER (there are the same number of BEFORE/AFTER's); so, every time a BEFORE table is printed, ALWAYS print an AFTER table

PROGRAM CODING AND SUBMISSION REQUIREMENTS

The instructors correctly running scheduler program may be seen by executing:

```
$ /opt/classdata/cs326/scheduler.exe
```

and using Ctrl-S to stop/start scrolling. Your program should produce similar results. Your results may not be exactly the same because you may use random numbers in a slightly different order. However, your program results must be in the SAME format, with the SAME blank line spacing, SAME text, EXACT SAME column headings, etc., and work correctly for 99 processes. The instructors program output can be seen by looking at the file:

```
/opt/classdata/cs326/scheduler.out
```

DO NOT, I repeat, DO NOT attempt to print the screen output from a scheduler run. This output is several HUNDRED pages long, will WASTE a tremendous amount of paper, and will incur the wrath of people you would rather have as your friends! You may save it to a file, but DO NOT print it. Make sure you UNDERSTAND the INSTRUCTORS OUTPUT COMPLETELY before you begin to write your own scheduler.

Your program must be fully documented, use correct C coding standards, and use the style taught in Data Structures and Algorithms. The process table should be an array of 10 process structures (or a linked-list), and may be global. All else is local. The main() function must contain the loop, and must call functions to do the program tasks. One should be able to look at main() and see what is being done, but most of the executing code should be contained in the called functions, as taught in your previous courses.

The beginning of the program must contain the program identification information taught in your Data Structures and Algorithms course, followed by the pledge from:

```
/opt/classdata/cs326/pledge.txt
```

UNCHANGED, which you have read, followed, and signed, indicating all code in this project is your own code, and none of it has been copied from anyone else. THIS IS "NOT" A GROUP PROJECT; YOU ARE "NOT" TO WORK TOGETHER. If you need help, you must see your instructor.

The FIRST function MUST be main(), followed by other functions, each properly documented, and each containing the code for only 1 task. Your program must be designed and coded so it can be easily modified in order to change the way it simulates the scheduling task, as taught in previous programming courses. While debugging, you may add scanf statements to stop the scrolling and wait for an Enter keypress, but in the final submitted version, the program must scroll continuously on the screen until PID 100 is reached. Each time the program is run, it is to produce the EXACT SAME output.

In your linux home directory, you MUST have a cs326 folder, and inside this folder, a scheduler folder. Failure to create and name these 2 folders exactly as follows:

```
/home/your_home_directory/cs326/scheduler  [Note: Use ALL Lower Case]
```

will cause your project NOT to be graded. Inside the scheduler folder, your project must be named scheduler.c. On or before the due date, you must submit a hard copy of scheduler.c, printed on an AC212 printer, from gedit, with these gedit options:

```
Edit/Preferences/View:   Check the first 3 boxes, UNcheck the last 3 boxes
Edit/Preferences/Editor: Tab width: 3;  Check Insert spaces...;  UNcheck Enable...
Edit/Preferences/Fonts & Colors: UNcheck Use the system fixed width font
View/Highlight Mode: Select Plain Text
File/Print/Text Editor:  Uncheck Print syntax highlighting;
                         Check Print line numbers
```

Clearly, in the upper right corner of the front page, print your name and id, CS326 (or CS326-1, -2, etc.), SCHEDULER PROJECT, and nn (your teacher assigned grading code), on 4 separate lines. Staple once in the upper left corner so every page is readable. Your submitted copy must have a pledge, signed in ink, UNCHANGED and OBEYED, with your name clearly PRINTED UNDER your signature. Your scheduler.c file must remain, UNCHANGED, in .../cs326/scheduler folder, until it has been graded. I will compile as follows:

```
$ gcc -Wall -ansi -oscheduler.exe scheduler.c -lm
```

and execute it, as well as look at the source code to determine your project grade. The 2 stress tests, as described previously, will also be performed on your scheduler.