4.8 (8 points) Which of the following components of program state are shared across threads in a multithreaded process.

(a) Register Values

(b) Heap Memory

(c) Global Variables

(d) Stack Memory

4.11 (5 points) Is it possible to have concurrency but not parallelism? Explain.

Yes. Parallelism requires multiple CPU's, so that multiple processes/threads are being run simultaneously in real time. Concurrency only requires that more than one process/thread is in the process of being computed at a time, thus, multiple process/thread may share a single CPU (not parallel) and alternate turns executing code by following some scheduling algorithm.

4.17 (10 points) The program shown below uses the pthreads API. What would be the output from the program at LINE C and LINE P?

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int value = 0;

void * runner(void * param); /* the thread */

int main(int argc, char ** argv) {
        pid_t pid;
        pthread_t tid;
```

```
        pthread_attr_t attr;
        pid = fork();

        if (pid == 0) { /* child process */
                pthread_attr_init(&attr);
                pthread_create(&tid, &attr, runner, NULL);
                pthread_join(tid, NULL);
                printf("CHILD: value = %d\n", value); /* LINE C */

        } else if (pid > 0) { /* parent process */
                wait(NULL);
                printf("PARENT: value = %d\n", value); /* LINE P */
        }

        return 0;
}

void * runner(void * param) {
        value = 5;
        pthread_exit(0);
}
```

C = 5

P = 0

6.14 (15 points) Consider the exponential average formula used to predict the length of the next CPU burst. What are the implications of assigning the values to the parameters used by the algorithm?

(a) $\alpha = 0$ and $\tau_0 = 100$ ms

(b) $\alpha = 0.99$ and $\tau_0 = 10$ ms

6.16 (30 points) Consider the following set of processes, with the length of the CPU burst given in milliseconds: Process Burst Time Priority

P1 2 2
P2 1 1
P3 8 4
P4 4 2
P5 5 3
The processes are assumed to have arrived in the order P1, P2, P3, P4, P5, all at time 0.

(a) Draw foru Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, non-preemptive priority (a larger priority number implies a higher priority), and RR (quantum = 2).

(b) What is the turnaround time of each process for each of the scheduling algorithms in part a?

(c) What is the waiting time of each process for each of these scheduling algorithms.

(d) Which of the algorithms results in the minimum average waiting time (over all processes)?

6.19 (5 points) Which of the following scheduling algorithms could result in starvation?

(a) First-Come, First-Served

(b) Shortest job first

(c) Round robin

(d) Priority

6.23 (7 points) Consider a preemptive priority scheduling algorithm based on dynamically changing priorities. Larger priority numbers imply higher priority. When a process is waiting for the CPU (in the ready queue, but not running), its priority change at a rate α. When it is running, its priority changes at a rate β. All processes are given a priority of 0 when they enter the ready queue. The parameters α and β can be set to give many different scheduling algorithms.

(a) What is the algorithm that results from $\beta > \alpha > 0$?

(b) What is the algorithm that results from $\alpha < \beta < 0$?

---

7 (20 points) Convert the following program to use threads. Under the following restrictions:

(a) One thread will print "hello", one thread will print "world", and the main function will print the trailing "\n", using just pthread˙create(), pthread˙exit(), pthread˙yield(), and pthread˙join().

(b) You must use a synchronization method to ensure the "world" thread runs after the "hello" thread.

(c) You must use a synchronization method to ensure that the main thread does not execute until after the "world" thread.

```c
#define _GNU_SOURCE
#include <pthread.h>
#include <stdio.h>
#include <sys/types.h>

// every thread will have access to these thread id's
// that way, I can have threads wait for (join with)
// any thread that I want
pthread_t pt_hello;
pthread_t pt_world;

void * world(void * param);
void * hello(void * param);

int main(int argc, char ** argv)  {
        // create the two threads
        pthread_create(&pt_hello, NULL, hello, NULL);
```

```
        pthread_create(&pt_world, NULL, world, NULL);

        // wait for world to finish (which in turn waits for hello)
        // before going on and printing the new line character
        pthread_join(pt_world, NULL);

        printf("\n");
        return 0;
}

void * world(void * param) {
        // wait for hello to finish before printing world
        pthread_join(pt_hello, NULL);
        printf("world");
        pthread_exit(0);
}

void * hello(void * param) {
        // hello doesn't have to wait for anything
        // it can print hello whenever it sees fit
        printf("hello ");
        pthread_exit(0);
}
```