Assignment 3
(**Due date: Thursday, 10/15/2009, in class**)

Your name:	Date:

Part One: Provide **brief** answers to the following Chapter Exercises questions**:**

4.7 Provide two programming examples in which multithreading does not provide better performance than a single-threaded solution

4.10 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

4.12 As described in Section 4.5.2, Linux does not distinguish between processes and threads. Instead, Linux treats both in the same way, allowing a task to be more akin to a process or a thread depending on the set of flags passed to the clone() system call. However, many operating systems—such as Windows XP and Solaris—treat processes and threads differently. Typically, such systems use a notation wherein the data structure for a process contains pointers to the separate threads belonging to the process. Contrast these two approaches for modeling processes and threads within the kernel.

4.14 Consider a multiprocessor system and a multithreaded program written using the many-to-many threading model. Let the number of user-level threads in the program be greater than the number of processors in the system. Discuss the performance implications of the following scenarios.

a. The number of kernel threads allocated to the program is less than the number of processors.
b. The number of kernel threads allocated to the program is equal to the number of processors.
c. The number of kernel threads allocated to the program is greater than the number of processors but less than the number of user-level threads.

5.9 Why is it important for the scheduler to distinguish I/O-bound programs from CPU-bound programs?

5.11 Consider the exponential average formula used to predict the length of the next CPU burst. What are the implications of assigning the following values to the parameters used by the algorithm?

a. $\alpha = 0$ and $\tau_0 = 100$ milliseconds
b. $\alpha = 0.99$ and $\tau_0 = 10$ milliseconds

5.12 Consider the following set of processes, with the length of the CPU-burst time given in milliseconds:

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

The processes are assumed to have arrived in the order P1, P2, P3, P4, P5, all at time 0.

a. Draw (**Hand drawing** OK) four Gantt charts illustrating the execution of these processes using FCFS, SJF, a nonpreemptive priority (a smaller priority number implies a higher priority), and RR (quantum = 1) scheduling.
b. (**Using a Table**) what is the turnaround time of each process for each of the scheduling algorithms in part **a**?
c. (**Using a Table**) what is the waiting time of each process for each of the scheduling algorithms in part **a**?
d. Which of the schedules in part a results in the minimal average waiting time (over all processes)?

5.13 Which of the following scheduling algorithms could result in starvation?

a. First-come, first-served
b. Shortest job first
c. Round robin
d. Priority

5.15 Consider a system running ten I/O-bound tasks and one CPU-bound task. Assume that the I/O-bound tasks issue an I/O operation once for every millisecond of CPU computing and that each I/O operation takes 10 milliseconds to complete. Also assume that the context switching overhead is 0.1 millisecond and that all processes are long-running tasks. What is the CPU utilization for a round-robin scheduler when:

a. The time quantum is 1 millisecond
b. The time quantum is 10 milliseconds

5.18 Explain the differences in the degree to which the following scheduling algorithms discriminate in favor of short processes:

a. FCFS
b. RR
c. Multilevel feedback queues

5.19 Using the Windows XP scheduling algorithm, what is the numeric priority of a thread for the following scenarios?

a. A thread in the REALTIME PRIORITY CLASS with a relative priority of HIGHEST.
b. A thread in the NORMAL PRIORITY CLASS with a relative priority of NORMAL.
c. A thread in the HIGH PRIORITY CLASS with a relative priority of ABOVE NORMAL.

5.20 Consider the scheduling algorithm in the Solaris operating system for time-sharing threads:

a. What is the time quantum (in milliseconds) for a thread with priority 10? With priority 55?
b. Assume a thread with priority 35 has used its entire time quantum without blocking. What new priority will the scheduler assign this thread?
c. Assume a thread with priority 35 blocks for I/O before its time quantum has expired. What new priority will the scheduler assign this thread?

5.21 The traditional UNIX scheduler enforces an inverse relationship between priority numbers and priorities: The higher the number, the lower the priority. The scheduler recalculates process priorities once per second using the following function:

Priority = (Recent CPU usage / 2) + base

where base = 60 and *recent CPU usage* refers to a value indicating how often a process has used the CPU since priorities were last recalculated.

Assume that recent CPU usage for process P1 is 40, process P2 is 18, and process P3 is 10. What will be the new priorities for these three processes when priorities are recalculated? Based on this information, does the traditional UNIX scheduler raise or lower the relative priority of a CPU-bound process?

599a. The program shown in the following uses the Pthreads API. What would be output from the program at **LINE C** and **LINE P**?

```
#include <pthread.h>
#include <stdio.h>

int value = 0;
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
      int pid;
      pthread_t tid;
      pthread_attr_t attr;

      pid = fork();

      pthread_attr_init(&attr);
      pthread_create(&tid,&attr,runner,NULL);
      pthread_join(tid,NULL);

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

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

599b. The program shown in the following uses the Pthreads API. What would be output from the program at **LINE C** and **LINE P**?

```
#include <pthread.h>
#include <stdio.h>

int value = 0;
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
      int pid;
      pthread_t tid;
      pthread_attr_t attr;

      pthread_attr_init(&attr);
      pthread_create(&tid,&attr,runner,NULL);
      pthread_join(tid,NULL);

      pid = fork();

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

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

599c. The program shown in the following uses the Pthreads API. What would be output from the program at **LINE C** and **LINE P**?

```
#include <pthread.h>
#include <stdio.h>

int value = 0;
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
      int pid;
      pthread_t tid;
      pthread_attr_t attr;

      pid = fork();

      if (pid == 0) {
            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 */
      }
}

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

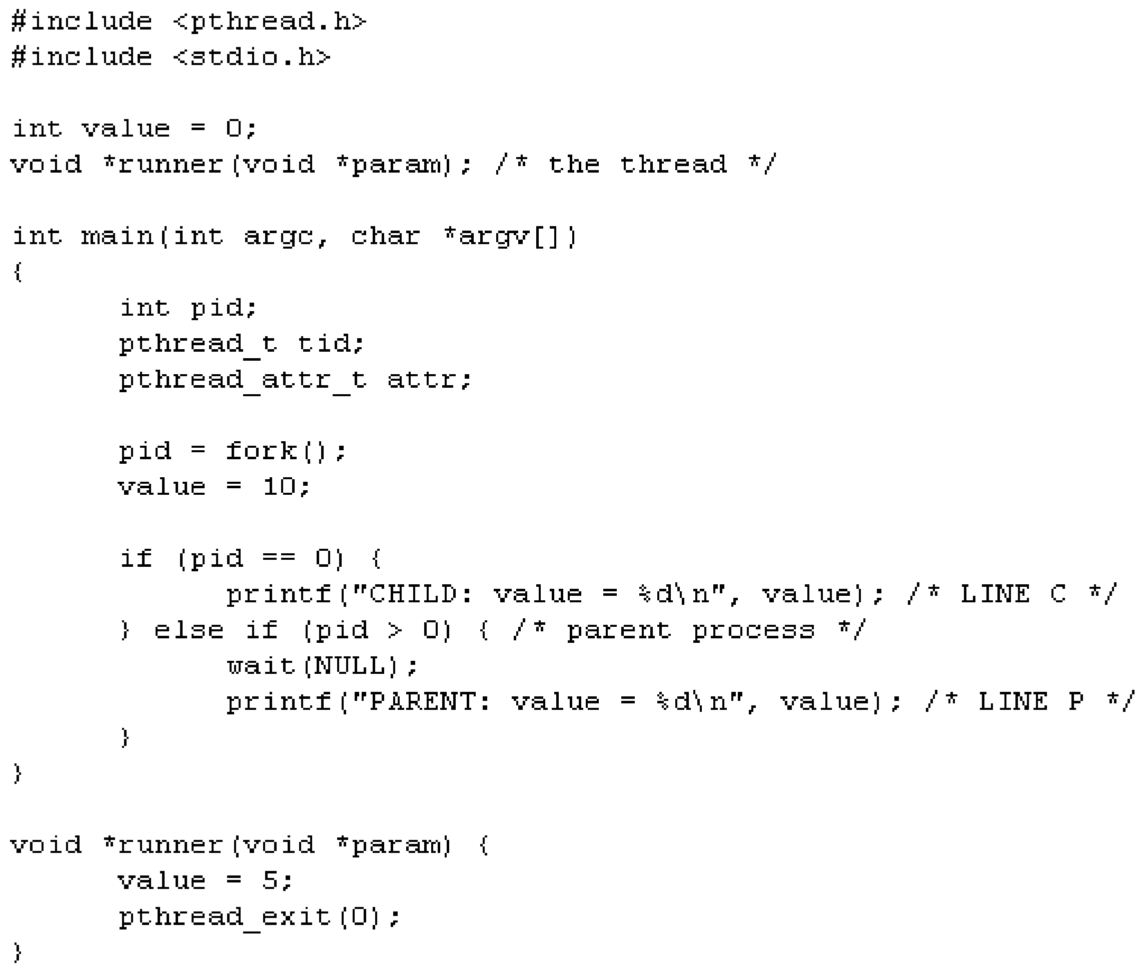
599d. The program shown in the following uses the Pthreads API. What would be output from the program at **LINE C** and **LINE P**?

```
#include <pthread.h>
#include <stdio.h>

int value = 0;
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
      int pid;
      pthread_t tid;
      pthread_attr_t attr;

      pid = fork();

      if (pid == 0) {
            printf("CHILD: value = %d\n", value); /* LINE C */
      } else if (pid > 0) { /* parent process */
            pthread_attr_init(&attr);
            pthread_create(&tid,&attr,runner,NULL);
            pthread_join(tid,NULL);
            wait(NULL);
            printf("PARENT: value = %d\n", value); /* LINE P */
      }
}

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

599e. The program shown in the following uses the Pthreads API. What would be output from the program at **LINE C** and **LINE P**?

```
#include <pthread.h>
#include <stdio.h>

int value = 0;
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
	int pid;
	pthread_t tid;
	pthread_attr_t attr;

	pid = fork();

	if (pid == 0) {
		value = 10;
		printf("CHILD: value = %d\n", value); /* LINE C */
	} else if (pid > 0) { /* parent process */
		pthread_attr_init(&attr);
		pthread_create(&tid,&attr,runner,NULL);
		pthread_join(tid,NULL);
		wait(NULL);
		printf("PARENT: value = %d\n", value); /* LINE P */
	}
}

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

599f. The program shown in the following uses the Pthreads API. What would be output from the program at **LINE C** and **LINE P**?



```
#include <pthread.h>
#include <stdio.h>

int value = 0;
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
      int pid;
      pthread_t tid;
      pthread_attr_t attr;

      pid = fork();
      value = 10;

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

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

599h. The program shown in the following uses the Pthreads API. What would be output from the program at **LINE C** and **LINE P**?

```
#include <pthread.h>
#include <stdio.h>

int value = 0;
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
      int pid;
      pthread_t tid;
      pthread_attr_t attr;

      pid = fork();
      value = 10;

      if (pid == 0) {
            printf("CHILD: value = %d\n", value); /* LINE C */
      } else if (pid > 0) { /* parent process */
            pthread_attr_init(&attr);
            pthread_create(&tid,&attr,runner,NULL);
            pthread_join(tid,NULL);
            wait(NULL);
            printf("PARENT: value = %d\n", value); /* LINE P */
      }
}

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

599i. The program shown in the following uses the Pthreads API. What would be output from the program at **LINE C** and **LINE P**?

```
#include <pthread.h>
#include <stdio.h>

int value = 0;
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
      int pid;
      pthread_t tid;
      pthread_attr_t attr;

      pid = fork();
      value = 10;

      if (pid == 0) {
            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 */
      }
}

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

Part Two: Important questions related to Chapters 4 & 5 **(No submission):**

- 4.1, 4.2, 4.3, 4.4
- 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7
- (check the course website for these questions and solutions)

=======================**Important Notes**===========================

- Solutions must be **typewritten**. You can use lists, bullets for the write-up (short phrases are OK; complete sentences *not* necessary).
- Put all your solutions **in the same order** as the above questions.
- Use this question paper as **cover page** and **staple them together**.
- *Electronic* submissions will be accepted ***only under*** an excusable circumstances (the above rules still apply)—in this case, put your solutions including this question paper as cover page into ***ONE SINGLE*** Word or PDF file and send it to me via email. I'll *grade your work based on this file* and send back only your grade (without corrections of your errors).
- The full score for this homework is **100** points. **You will lose 5-10 points for missing ANY ONE of the followings in your submission:**
    - No name
    - No cover page
    - Your work submitted *not* in proper order
    - Not a single Word or PDF file (if submitted via email)

==