# x Homework #2 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_

Grading: 3 = correct

2 = almost

1 = an attempt

0 = nothing

Score: Points / Possible

# (55 points) (Name) (Section)

**Chapter 3 – Processes**

**Chapter 4 – Threads**

|  |  |
| --- | --- |
| Questions: | Answers: |
| 1. (5 points) Consider a concurrent program with two processes, *p* and *q*, defined as follows: A, B, C, D, and E are arbitrary atomic (indivisible) statements. Assume that the main program (not shown) does a **parbegin** of the two processes (starts them both executing). Show all the possible interleaving of the execution of the preceding two processes.  **void p() void q()**  **{ A; { D;**  **B; E;**  **C; }**  **}** | ABCDE, ABDCE, ABDEC, ADBCE, ADBEC, ADEBC,  DABCE, DABEC, DAEBC, DEABC |
| 2. (10 points) Explain how the following applications would benefit from multithreading:   1. Web browser 2. Word processor 3. Multicore system 4. Operating system kernel | a. Can process multiple tabs, can use multiple threads for multiple tabs. Can use a thread for each tab.b. Threads can different jobs. One could be printing what you type, one could be spell checking, one could be taking down statistical data an so on. You could also have multiple threads working each job.c. Each thread could run a different core and run concurrently.d. Operating system kernel would benefit by multithreading by utilizing multiple cores, allowing for processes to run concurrently. |
| 3. (10 points) Explain the benefits of multithreaded programming with respect to:   1. Responsiveness 2. Resource sharing 3. Economy 4. Scalability | a. Responsiveness would improve with multithreading because a task on one thread is not dependent on another thread to finish first. They can do their jobs concurrently thus improving responsiveness.b. Resources can be shared between threads but can’t share resources concurrently. One thread could use a resource while others are waiting for I/O.c. Multithreading would limit resource waste. It will also save time which will thus save money and energy of the process.d. You will be able to add and subtract threads without deterring other processes. |
| 4. (15 points) Elaborate on the programming challenges of multithreading applications.  a. Identifying units of work (tasks)  b. Balance  c. Data splitting  d. Data dependency  e. Testing and debugging | a. Tasks can be threaded or sequential.b. It could be difficult to make sure each thread is being executed at the proper time. Threads can have different priorities are executed sequentially of those priorities. It may be difficult to enforce this. It may be difficult to switch between them or run concurrently.c. Some date is unique to one task on one thread while sometimes multiple threads will share the same date. Splitting the data between threads can be very difficult to program. Splitting data that is grouped together between different tasks could be very difficult but not doing so could be wasteful.d. Multiple tasks will need to access the same memory resulting in concurrency and dead lock problems.e. Since multiple threading is not sequential, it is difficult to trace where the problem occurred. |
| 5. (15 points) The output of the C program to the right is:  **tid=0, count=1**  **tid=1, count=1**  **tid=2, count=1**  **tid=3, count=1**  **tid=0, count=2**  **tid=1, count=2**  **tid=2, count=2**  **tid=3, count=2**  Answer the following questions:  a) Where in memory would you find the variable **i** (heap, kernel stack, thread stack)?  Kernel stack  b) Why is the variable **count** equal to 1 for the first four **printf**‘s in function **myThread** and then changes to 2?  Each thread goes through one iteration and prints off count one time. Count is at One for each threads first iteration. Then when it goes through each threads second iteration, count is incremented in each thread, and so each thread will print off 2 sequentially.  c) What is the purpose of the **setjmp** function on line 025?  checks to see if the current thread is blocked. If it’s not blocked it calls longjmp(kernel, 1);  d) How many times is the function **myThread** called? What is the value of **code** when the function is called?  MyThread is called 4 times.  Code will be two.  e) What C statement is executed just after line 041 is executed for the 5th time?  Line 42: printf | **001 #include <setjmp.h>**  **002 #include <stdio.h>**  **003 #include <stdlib.h>**  **004 #include <ctype.h>**  **005**  **006 #define NUM\_THREADS 4**  **007 #define STACK\_SIZE (64\*1024)**  **008 #define STACK\_END (STACK\_SIZE/sizeof(int\*))**  **009 volatile void\* stack; // stack**  **010**  **011 int tid; // thread id**  **012 jmp\_buf thread[NUM\_THREADS]; // thread context**  **013 jmp\_buf kernel; // kernel context**  **014 void myThread(int); // thread function**  **015**  **016 int main()**  **017 {**  **018 int i, code;**  **019 for (tid = 0; tid < NUM\_THREADS; tid++)**  **020 {**  **021 if (setjmp(kernel) == 0)**  **022 {**  **023 stack = (int\*)malloc(STACK\_SIZE) + STACK\_END;**  **024 \_asm("movl \_stack,%esp"); // new stack pointer**  **025 if (!(code = setjmp(thread[tid]))) longjmp(kernel, 1);**  **026 myThread(tid);**  **027 }**  **028 }**  **029 for (i = 0; i < 12; i++)**  **030 {**  **031 tid = i % NUM\_THREADS; // select next thread**  **032 if (!(code = setjmp(kernel))) longjmp(thread[tid], 2);**  **033 }**  **034 }**  **035**  **036 void myThread(int tid)**  **037 {**  **038 int count = 0; // task iteration counter**  **039 while (1)**  **040 {**  **041 if (!setjmp(thread[tid])) longjmp(kernel, 3);**  **042 printf("\ntid=%d, count=%d", tid, ++count);**  **043 }**  **044 }** |