CSI3131 – Operating Systems
Tutorial 3 – Threads - Solution

1. What are two differences between user-level threads and kernel-level threads? Under what circumstances is one type better than the other?

   (1) User-level threads are unknown by the kernel, whereas the kernel is aware of kernel threads. (2) On systems using either M:1 or M:N mapping, user threads are scheduled by the thread library and the kernel schedules kernel threads. (3) Kernel threads need not be associated with a process whereas every user thread belongs to a process. (4) Kernel threads are generally more expensive to maintain than user threads as they must be represented with a kernel data structure.

2. What resources are used when a thread is created? How do they differ from those used when a process is created?

   Because a thread is smaller than a process, thread creation typically uses fewer resources than process creation. Creating a process requires allocating a process control block (PCB), a rather large data structure. The PCB includes a memory map, list of open files, and environment variables. Allocating and managing the memory map is typically the most time-consuming activity. Creating either a user or kernel thread involves allocating a small data structure to hold a register set, stack, and priority.

3. Describe the relationship between user and kernel threads for each of the following models: Many-to-One, One-to-one, and Many-to-Many. In particular, indicate where scheduling is done in each of these models.

   Many to one:
   - **Threads are manages in the process by a threads library that is run in user space. The kernel is unaware of the threads.**
   - **Thread scheduling: done by the threads library in user space.**
   - **Advantage: Efficient since does not involve the kernel in thread scheduling, number of threads limited by developer of thread library and application.**
   - **Disadvantages: When a thread blocks on a system call made to the kernel, all threads in the process will be blocked. Cannot run threads in the same process on separate CPUs**

   One to one:
   - **OS is now aware of threads in the process. Each user thread is mapped to a kernel thread.**
   - **Scheduling: done by kernel**
   - **Advantage: a blocking thread no longer blocks other threads from execution, threads can run on different CPUs**
   - **Disadvantage: Kernel is involved in management which increases overhead. The number of threads allowed in a process is normally limited by OS.**

   Many to many:
   - **A number of kernel threads are allocated to a process for supporting the execution of user threads. User threads are then scheduled to run overtop kernel threads (LWP) seen as virtual CPUs.**
   - **Scheduling: between a process thread library and the kernel.**
   - **Advantage: Many user threads can be created while concurrency is supported with kernel threads.**
   - **Disadvantage: Difficulty in coordinating scheduling between user thread library and kernel.**

4. The program shown below uses Pthreads. What would be output from the program at LINE C and at LINE P? Explain your answer.

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

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

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

   pid = fork();

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

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

Line C: CHILD: value = 5 **since the thread changes the value of the global variable.**

Line P: PARENT: value = 0 **no change to the global variable is made in the parent, changes made in the child process does not affect the global variables in the parent process.**

5. Consider that two processes start at time t=0. Process A operates with a single thread, while Process B runs a multi-threading program that initiates three threads. Thread T1 starts at time t=0, thread T2 starts after T1 has run (i.e. assigned to the CPU) for 19 time units (t.u.), and thread T3 starts after T1 has run for 27 t.u. (that is, T1 invokes the T2 and T3 threads). Process A and each of the threads of Process B run a series of CPU bursts separated by I/O operations requested from the OS. Assume that each I/O operation takes 52 time units to complete. The execution of Process A and each thread of Process B occur as follows:

   Process A (no multithreading used)

      55 t.u. CPU burst, I/O request, 50 t.u. CPU burst, I/0 request, 30 t.u. CPU burst, I/O request

   Process B (multi-threaded)

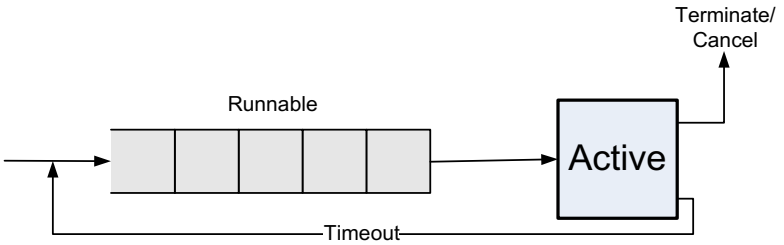
      T1: 300 t.u. CPU burst (CPU bound thread)

      T2: 12 t.u. CPU burst, followed by I/O request (repeated 5 time) (I/O bound thread)

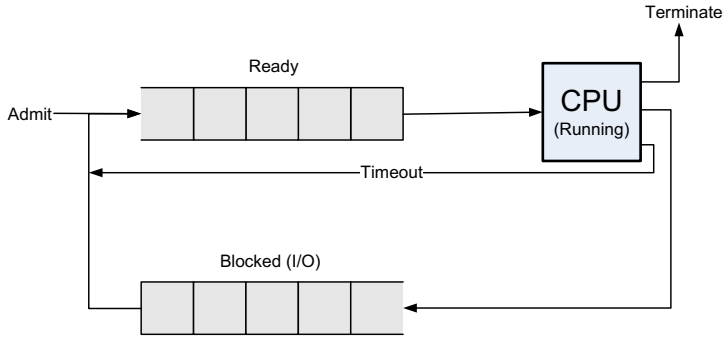
      T3: 29 t.u. CPU burst, I/O request, 10 t.u. CPU burst, I/0 request, 40 t.u. CPU burst, I/O request

   To simplify the problem, assume that the time for running OS and Threading Library management code as negligible (i.e. time to switch threads or switch processes shall be consider to be 0).

a) **Many to one model**: First consider that multi-threaded processes are supported by the computer system using a many to one model, that is, a thread library is responsible for managing threads in a process. Assume that the thread library uses two states (Runnable and Active) and a single scheduling queue as shown below for managing threads. A thread is moved to the Runnable state when it is waiting to become active and the Active state when it is allowed to execute (essentially assigned to the kernel thread). Typically other states are defined for managing user threads such as Sleeping and Stopped. But for simplicity, these states are not included for this problem. Assume that the Thread Library allows a threads maximum of 5 time units in the Active state before switching to another thread.



The OS on the other hand uses three states and two scheduling queues shown below. The Ready, Blocked, and Running states are used to represent processes ready to run, blocked waiting for an I/O operation to complete, and allocated to the CPU respectively. Note that states such as New, Terminated, and states for suspended processes are not included, again to keep the problem simple. Assume that the OS assigns the CPU to a process for a maximum of 10 time units, after which the CPU is allocated to the next process in the Ready queue.



Assume that both the threading library and the OS assign the Active/Running state to the thread/process at the head of the Runnable/Ready queues. Any process/thread added to these queues are placed at the end of the queue. This is a first-in first-out scheduling algorithm (this algorithm along with other scheduling algorithms shall be studied shortly in Module 4).

Complete the following table to show how processes and threads move between queues and the Active/Running state up to time 200 t.u. If a scheduling event at both user and kernel level occur at the same time, then assume that only the OS scheduling occurs, and that the user thread library will complete its scheduling operation the next time the CPU is assigned to the process (see time 50 for an example). Recall that the thread library allows a thread to be in the Active state for a maximum of 5 time units while processes are allocated to the CPU (i.e. in the running state) for a maximum of 10 time units. Also assume that time progresses for threads only when the process B is in the Running state, that is, allocated to the CPU. This means that when a thread is in the Active state and the process is in either the Ready or Blocked state, the threading library does not consider that time increases for the thread in the Active state.

	CPU	**Operating System**		**Process B**	
Time	Running	Ready queue	Blocked queue	Active	Runnable Queue
0	A(0)*	B(0)		T1(0)	
10	B(0)	A(10)		T1(0)	
20	A(10)	B(10)		T1(10)	
30	B(10)	A(20)		T1(10)	
39	B(19)	A(20)		T1(19)	T2(0)
40	A(20)	B(20)		T1(20)	T2(0)
50	B(20)	A(30)		T2(0)	T1(20)
55	B(25)	A(30)		T1(20)	T2(5)
60	A(30)	B(30)		T1(25)	T2(5)
70	**B(30)**	**A(40)**		**T2(5)**	**T1(25)**
75	**B(35)**	**A(40)**		**T1(25)**	**T2(10)**
77	**B(37)**	**A(40)**		**T1(27)**	**T2(10),T3(0)**
80	**A(40)**	**B(40)**		**T1(30)**	**T2(10),T3(0)**
90	**B(40)**	**A(50)**		**T2(10)**	**T3(0), T1(30)**
92	**A(50)**		**B(42)**	**T2(12)**	**T3(0), T1(30)**
97			**B(42), A(55)**	**T2(12)**	**T3(0), T1(30)**
144	**B(42)**		**A(55)**	**T2(12)**	**T3(0), T1(30)**
147	**B(45)**		**A(55)**	**T3(0)**	**T1(30), T2(15)**
149	**B(47)**	**A(55)**		**T3(2)**	**T1(30), T2(15)**
152	**B(50)**	**A(55)**		**T1(30)**	**T2(15), T3(5)**
154	**A(55)**	**B(52)**		**T1(32)**	**T2(15), T3(5)**
164	**B(52)**	**A(65)**		**T1(32)**	**T2(15), T3(5)**
167	**B(55)**	**A(65)**		**T2(15)**	**T3(5), T1(35)**
172	**B(60)**	**A(65)**		**T3(5)**	**T1(35), T2(20)**
174	**A(65)**	**B(62)**		**T3(7)**	**T1(35), T2(20)**
184	**B(62)**	**A(75)**		**T3(7)**	**T1(35),T2(20)**
187	**B(65)**	**A(75)**		**T1(35)**	**T2(20), T3(10)**
192	**B(70)**	**A(75)**		**T2(20)**	**T3(10), T1(40)**
194	**A(75)**	**B(72)**		**T2(22)**	**T3(10), T1(40)**
204	**B(72)**	**A(85)**		**T2(22)**	**T3(10), T1(40)**

Process A scheduled first by OS
T1 only active thread at t=0, assigned to active state by thread library

OS switches between processes every 10 time units

T2 arrives after T1 has run for 19 time units and is placed in the runnable queue

Thread library switches between threads every 5 time units of execution on the CPU

T2 arrives after T1 has run for 27 time units

Process B is blocked when T2 requests I/O

Process A is blocked when it requests I/O

Process B is reassigned to the CPU when the I/O operation is complete

Process A placed back in the ready queue when its I/O operation is complete.

* - the number in parentheses beside the A, B, T1, T2 and T3 represents the relative execution time (the time the process/thread executes on the CPU). Note that these times are incremented only when the process/thread is allocated to the CPU.

b) **One to one model:** Now consider that the OS is aware of the threads and applies the three states not only to processes but also to threads in a multi-threaded process. No threading library is involved in the scheduling and management of the process threads. Assume that the OS allocates the CPU to a process/thread a maximum of 10 time units before switching the CPU to another processes/thread. Complete the following table to show how threads in process B and process A move between the Ready and Blocked queues and the Running state. As before assume a first-in, first-out queuing discipline for the Ready and Blocked queues. Like with case (a), assume that T2 arrives after T1 has executed 19 time units and T3 arrives after T1 has executed for 27 time units (i.e. it is T1 that invokes the threads T2 and T3).

	CPU	**Operating System**	
Time	Running	Ready queue	Blocked queue
0	A (0)	T1 (0)	
10	T1 (0)	A (10)	
20	A (10)	T1 (10)	
30	T1 (10)	A (20)	
39	T1 (19)	A (20), T2 (0)	
40	A (20)	T2 (0), T1 (20)	
50	T2 (0)	T1 (20), A (30)	
60	T1 (20)	A (30), T2 (10)	
67	**T1 (27)**	**A (30), T2 (10), T3 (0)**	
70	**A (30)**	**T2 (10), T3 (0), T1 (30)**	
80	**T2 (10)**	**T3 (0), T1 (30), A (40)**	
82	**T3 (0)**	**T1 (30), A (40)**	**T2 (12)**
92	**T1 (30)**	**A (40), T3 (10)**	**T2 (12)**
102	**A (40)**	**T3 (10), T1 (40)**	**T2 (12)**
112	**T3 (10)**	**T1 (40), A (50)**	**T2 (12)**
122	**T1 (40)**	**A (50), T3(20)**	**T2 (12)**
132	**A(50)**	**T3(20), T1(50)**	**T2(12)**
134	**A(52)**	**T3(20), T1(50), T2(12)**	
142	**T3(20)**	**T1(50), T2(12), A(60)**	
151	**T1(50)**	**T2(12), A(60)**	**T3(29)**
161	**T2(12)**	**A(60), T1(60)**	**T3(29)**
171	**A(60)**	**T1(60), T2(22)**	**T3(29)**
181	**T1(60)**	**T2(22), A(70)**	**T3(29)**
191	**T2(22)**	**A(70), T1(70)**	**T3(29)**
193	**A(70)**	**T1(70)**	**T3(29), T2(24)**
201	**T1(70)**	**A(70)**	**T3(29), T2(24)**
203	**T1(72)**	**A(70), T3(29)**	**T2(24)**

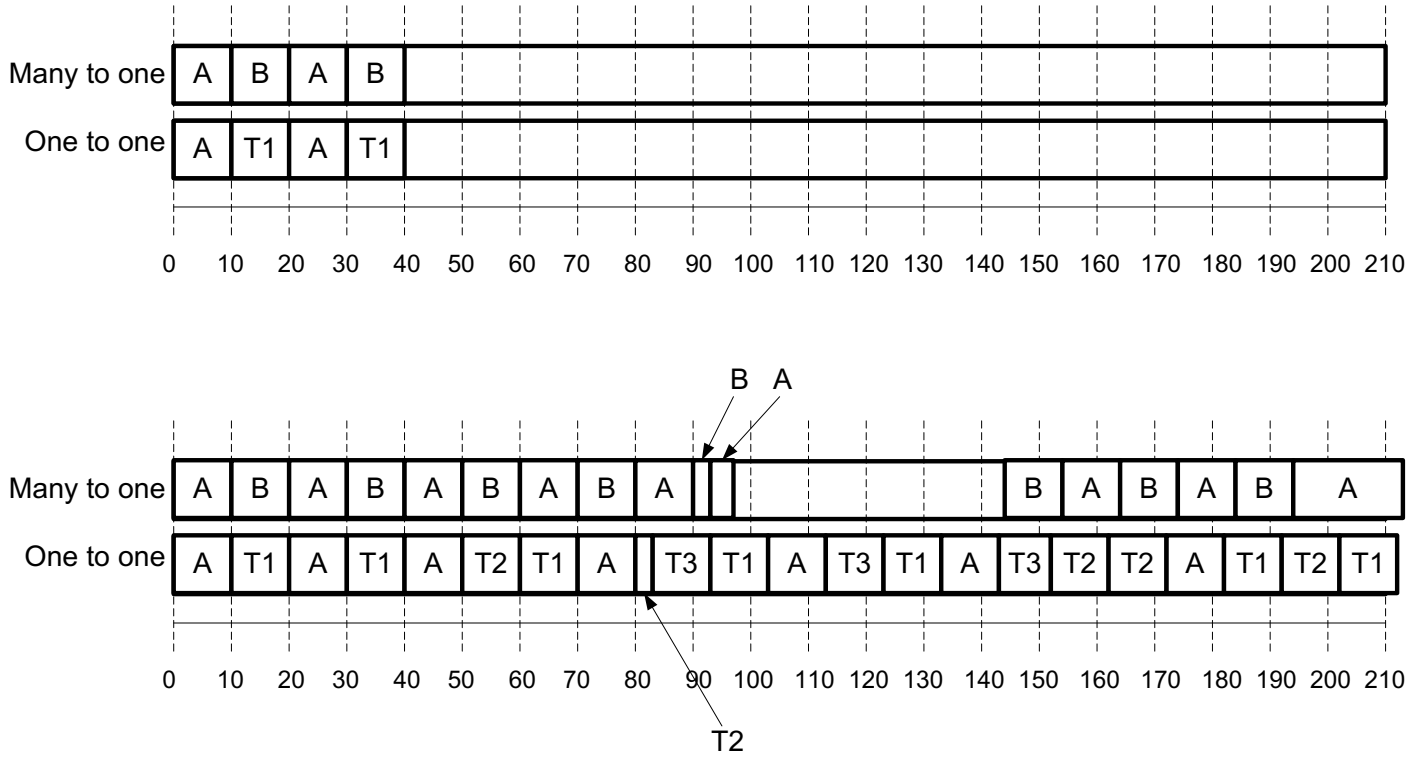
Process A scheduled first by OS T1 , the only active thread at t=0, is placed in the ready queue.

Thread T2 is created after T1 has run for 19 t.u. and is placed in the ready queue.

T2 is placed in the Blocked queue when it requests I/O

T2 is placed back in the ready queue when its I/O operation is complete.

c) On the following diagram, show which processes or threads are allocated to the CPU for both cases of (a) and (b). Note that the perspective presented is from the point of view of the OS – that is, in the case of the many to one model, indicate which processes are allocated to the CPU while in the case of the one to one model, show what threads of process B are allocated to the CPU. Comment on the difference in concurrency between the two models.



The one to one model provides better concurrency since the CPU is occupied during the whole time period. This is due to the fact that the OS can run other threads when the one to one model is used. In the many to one model, a blocking thread blocks the whole process, that is, all other threads in the process.