COMP 4735: Operating Systems

Lecture 7: Threads

Rob Neilson

rneilson@bcit.ca

Reading

- The following sections should be read before next Monday
  - it would also help to read this before your lab this week
  - Textbook Sections: 2.2 (Theory Part)
    10.3.3 (Case Study Part)
- Yes, there will be a quiz next Monday in lecture on the above sections
  - A sample quiz will be posted on webct on Friday
  - This quiz (Quiz 4) covers material from all of the sections listed above

Agenda

Key concepts for this lesson:

- What is a thread?
- Why do we need threads?
- How do threads work?
- The classic thread model.
- Multi-threaded programming examples.

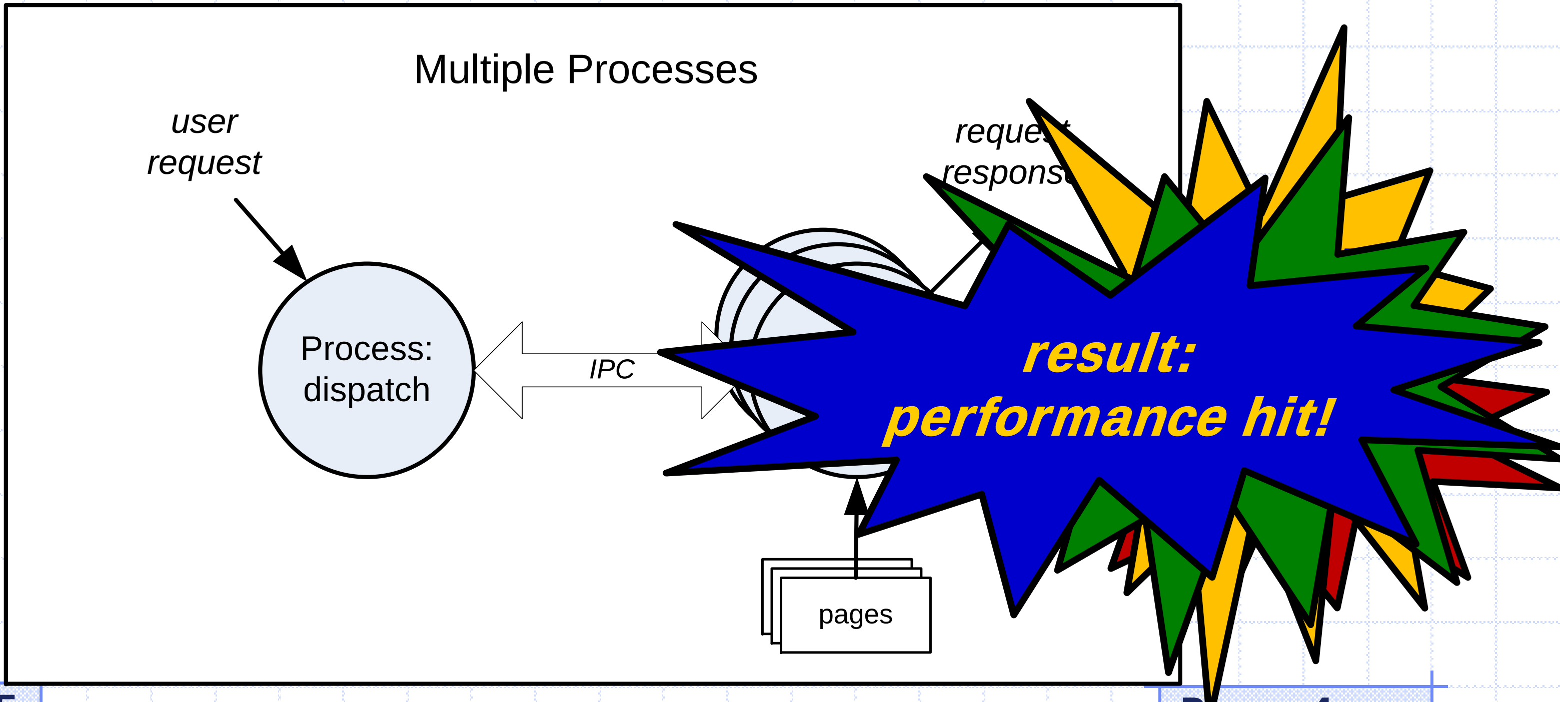
The Idea of Cooperating Processes

```
while (TRUE) {
    get_next_request(&buf);
    handoff_work(&buf);
}
```

(a)

```
while (TRUE) {
    wait_for_work(&buf)
    look_for_page_in_cache(&buf, &page);
    if (page_not_in_cache(&page))
        read_page_from_disk(&buf, &page);
    return_page(&page);
}
```

(b)



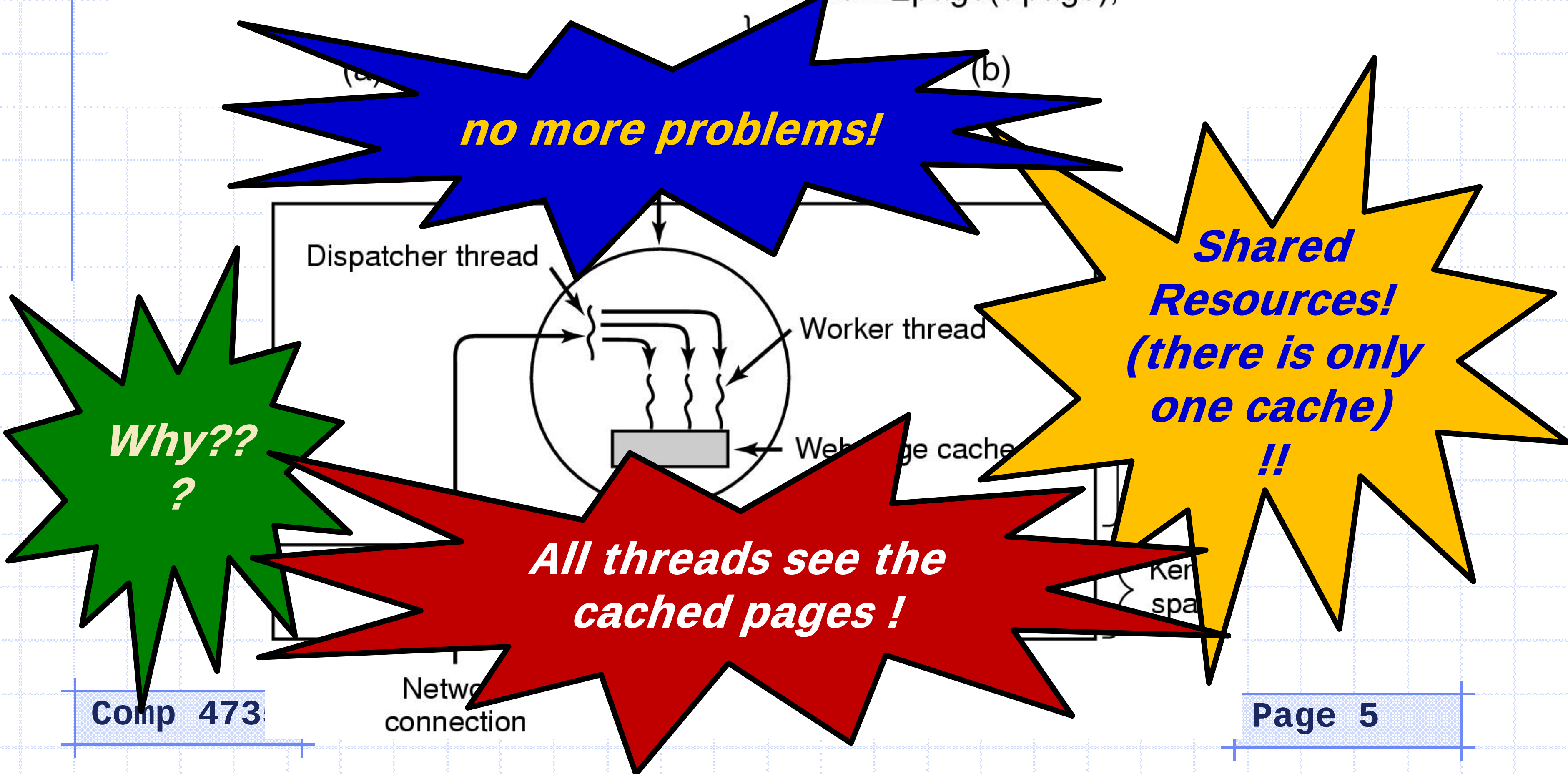
Another Approach: Cooperating Threads

```
while (TRUE) {
      get_next_request(&buf);
      handoff_work(&buf);
}
```

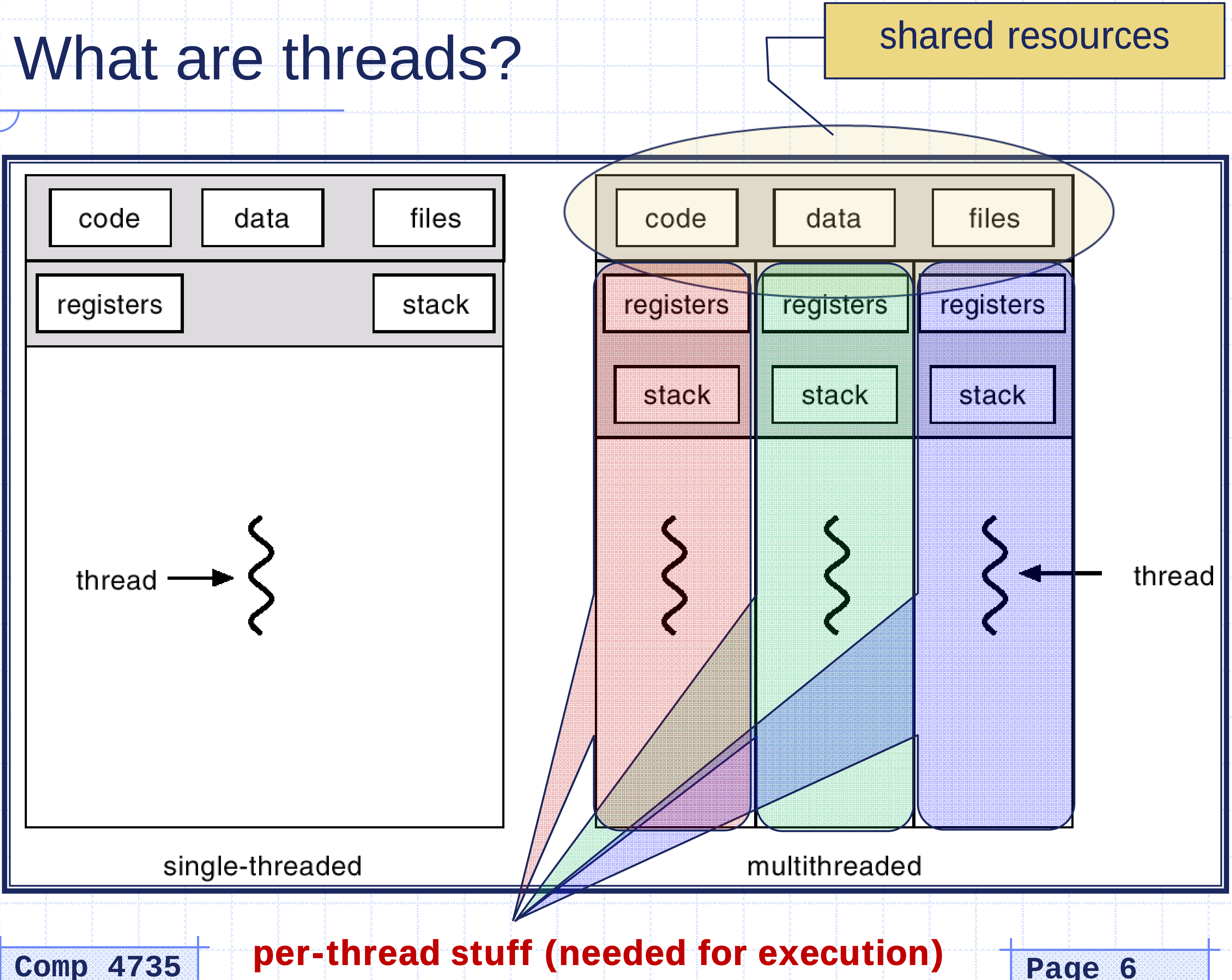
(a)

```
while (TRUE) {
      wait_for_work(&buf)
      look_for_page_in_cache(&buf, &page);
      if (page_not_in_cache(&page))
            read_page_from_disk(&buf, &page);
      return_page(&page);
}
```

(b)



What are threads?



Why threads - 1?

1. faster create/destroy
   - process creation takes time
     - need to create an address space
     - allocate blocks of memory
     - write or copy values into memory (possibly from disk)
     - context switch is required each time we create a process
   - *if we can reduce the creation time, application performance will be faster*
   - **threads achieve this because they do not have to create a complete address space and process table each time**

Why threads - 2?

2. faster context switch
   - consider multiple cooperating processes
   - they require numerous context switches to work together
   - context switches are expensive from an OS point of view
     - need to save entire state
     - need to load a new context including address space
     - need to run the scheduler
   - *it would be much quicker if we could keep the same context loaded and just switch the execution stack and PC registers*
   - **threads achieve this because control can be passed between threads without performing a compete context switch**

Why threads - 3?

3. increased parallelism (... pseudo-parallelism)
   - consider cooperating processes
     - they need to share information to work together
     - if IO is required, the process must block
   - traditionally we use IPC to achieve cooperation (pseudo-parallelism)
     - need to formulate/send a message, or
     - need to make a system call to use a pipe or socket or whatever
     - need to design carefully to ensure IPC works
   - *it would be much easier if we could just define a few variables and have them shared between the units of execution (threads)*
   - **threads achieve this because all the threads in a process share the same address space, so they can define global variables that are visible to all threads**

Why threads - 4?

- Summary of benefits of threads:
  - increased performance of the system
  - increased pseudo-parallelism within a process

Question 1

1. Which one of the following programs would not see an increase in performance through the use of multiple threads?
   a) an application that performs a lot of IO
   b) an application that uses mostly CPU and very little IO
   c) an application that is already designed for multiple processes
   d) an application that does not have a GUI (ie: a console app)
   e) all of the above would benefit from using multi-threading

Question 2

2. Which one of the following is an advantage of using threads?
   a) global variables are easier to code and are supported better
   b) threads support multiple instances of a program (such as a window) whereas processes do not
   c) threads allow for a shared program stack whereas processes do not
   d) a program can continue to execute while waiting for IO
   e) none of the above is an advantage

Traditional Process Model

Conceptually, the Process Model is based on two concepts:

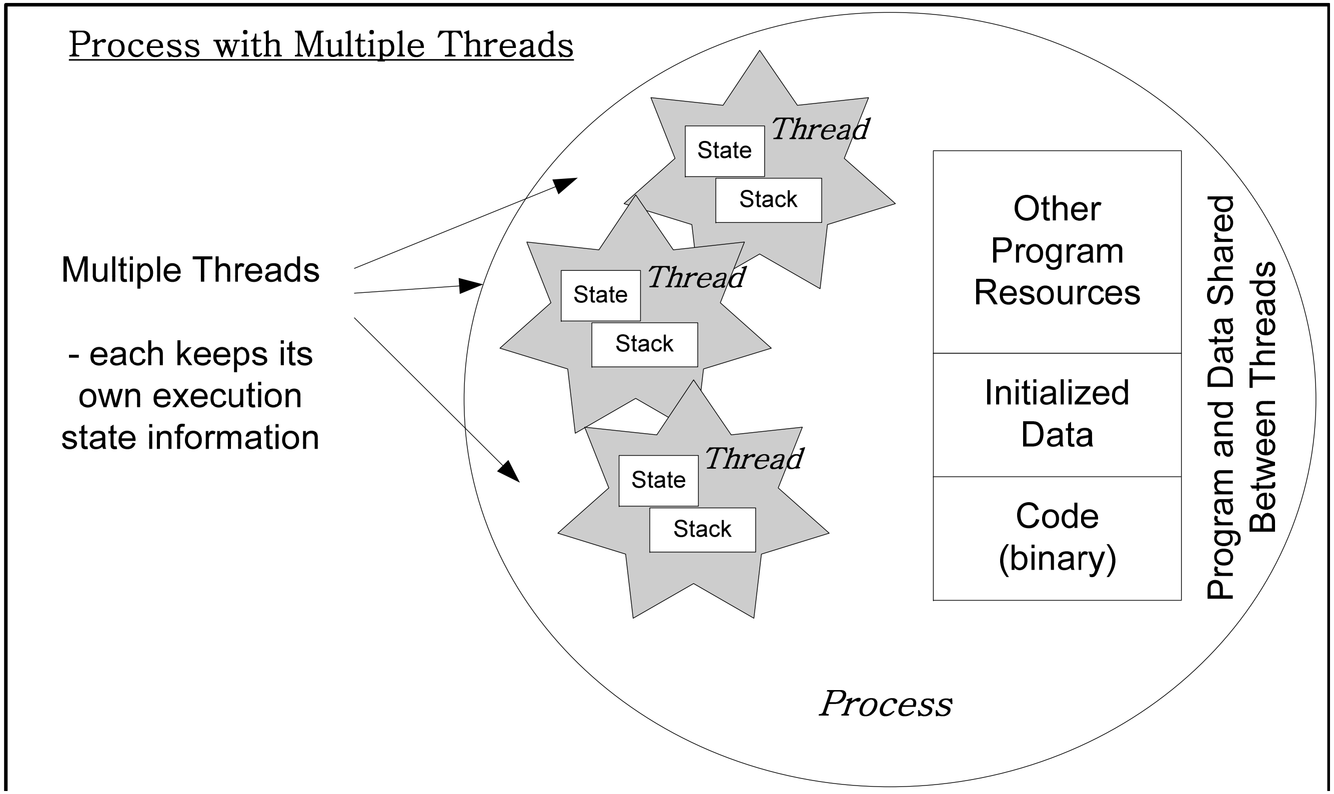
1. resource grouping
   - resources are *things that a process uses*, such as
     - program text, data, files, alarms, etc.
   - these items are grouped together and allocated to the process

non-multi-threaded

2. execution
   - each traditional process has a single thread of execution
   - this thread of execution defines the place in the program that is running, as well as the *current state of execution*, specifically
     - next instruction (PC), register values, history or procedure calls (stack) etc

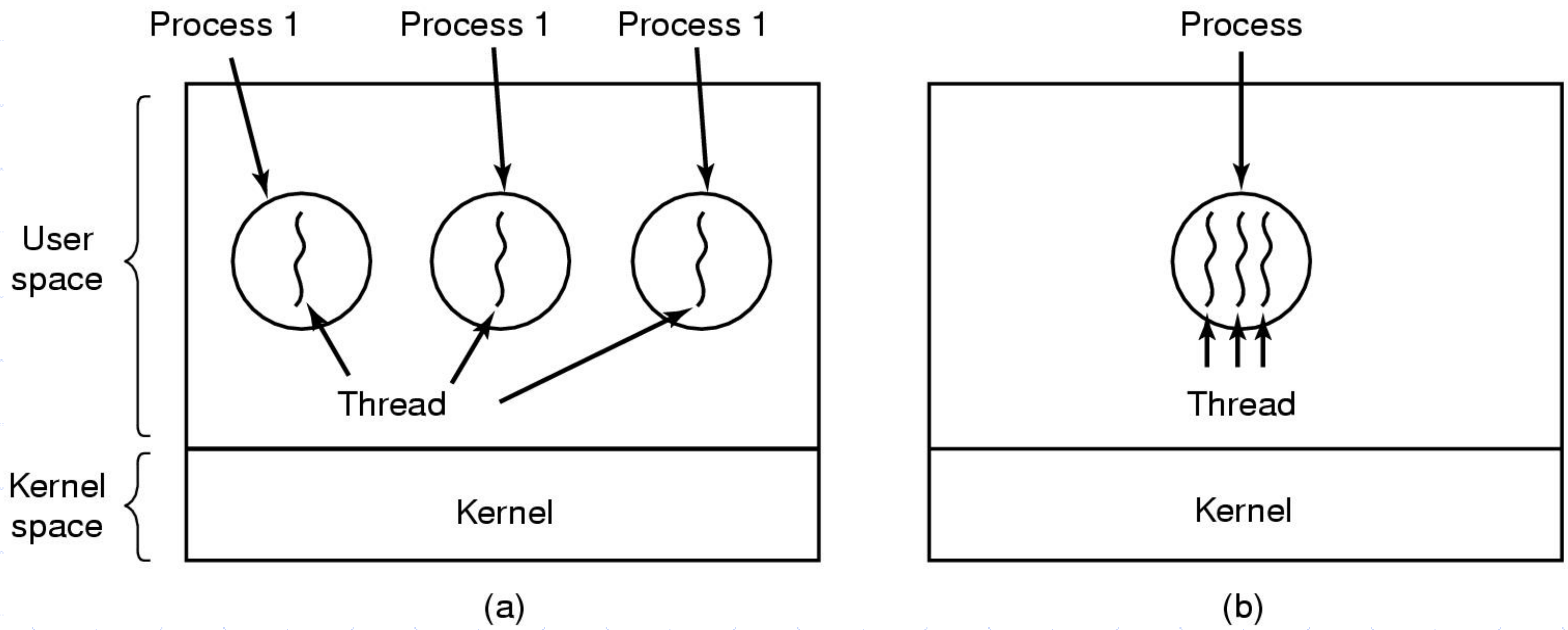
Thread Model (1)

- **Multi-threading: multiple threads in one process**



- **How?**
  - ***Separate the stuff that is needed for execution from the resources***
  - ***Allow multiple threads of execution to exist and operate on a single group of resources***

Thread Model (2)



- The process concept still exists in the Thread Model
  - *process defines the environment* (address space, program, files, etc)

- The thread(s) are created as needed to execute the program(s)
  - *threads define the execution state*(s)

What to put in the thread?

Per process items	**Per thread items**
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

- the OS keeps separate tables for process specific items and thread specific items
- in other words, each process now has:
  - one process table
  - many thread tables (one for each thread of execution)

Facts about the Thread Model

- every process has at least one thread
- threads are scheduled for execution
  - each thread has its own execution stack, state information
- each thread has a state transition model
  - threads can be in different states from each other
- threads block independently
  - one blocked thread does not stop the other threads in the process
- execution switches back and forth between threads, not processes
- all threads in a process share an address space, resources, global variables
- no protection between threads
  - all threads can read and write the same memory locations
  - this means that one thread could destroy another threads work

Question 3

3. Assume that a multi-threaded application is blocked waiting to write to the parallel port. Which one of the following statements is true?

   a) all threads in the process are blocked
   b) at least one thread in process is blocked
   c) at most one thread in the process is blocked
   d) the process is blocked but the threads continue to execute
   e) the write system call is non-blocking, so nothing is blocked

Question 4

4. Assume that you compile and run the helloworld.c program on a multi-threaded OS. How many threads are created?
   a) zero
   b) one
   c) two
   d) four
   e) there is not enough information to answer the question

```
#include <stdio.h>
int main(int argc, char* argv[]) {
   printf("Hello, world!\n");
   return 0;
}
```

Thread Related System Calls

- We are going to need some way to manage threads if we are to write multi-threaded applications.
- Typically we will have:
  - thread_create()
    - create a new thread in the current process
    - causes a new stack and thread table to be initialized
    - typically you specify the procedure to run in the thread (ie: to initialize the threads program counter to the desired entry point in the code)
  - thread_exit()
    - terminate the thread
    - other threads in the process will continue to run

Other Thread Related System Calls

- Sometimes we want to synchronize the actions of threads. The following system calls allow us to do this:
  - thread_join()
    - the thread running this command will block and wait for a specified thread to exit
    - for example, maybe this is a thread that waits for a window to be closed before it allows another one to be created
  - thread_yield()
    - give up CPU and let another thread run
    - this command exists to enable sharing between threads
    - most useful for CPU bound threads that need to cooperate with other thread sin the same process
    - only really makes sense for user-space threads, which we will discuss shortly

Sample Multi-threaded Program (C / Pthreads)

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

void *t_hello_world (void *tid) {
   printf ("Hello world - I am thread %d\n", tid);
   pthread_exit(0);
}

int main (int argc, char *argv[])
{
   pthread_t threads[10];
   int rc, i;
   for (i=0; i < 10; i++) {
      printf("In main. Creating thread %d\n",i);
      rc = pthread_create(&threads[i], 0, t_hello_world, (void *)i);
      if (rc != 0) {
         printf ("pthread_create: error code %d\n", rc);
         exit(-1);
      }
   }
   exit(0);
}
```

Sample Multi-threaded Program (java)

```
public class Hello implements Runnable {

   private int myNumber;

   public Hello(int number) {
      myNumber = number;
   }

   public void run() {
      System.out.println("Hello from thread " + myNumber);
   }

   public static void main(String args[]) {

      Thread threads[] = new Thread[10];

      for (int i=0; i<10; i++) {
         threads[i] = new Thread(new Hello(i));
         threads[i].start();
      }
   }
}
```

Question 5

5. Assume that you compile and run the program shown below on a multi-threaded OS. How many threads are created?
   a) zero
   b) one
   c) two
   d) three
   e) four

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

void *bye (void *id) {
   printf ("I am about to die ... %d\n", id);
   pthread_exit(0);
}

int main (int argc, char *argv[])
{
   int i=1;
   int rc;
   pthread_t threads[3];
   rc=pthread_create(&threads[i], 0, bye, (void *)i);
   exit(0);
}
```

Question 6

6. Which of the following Pthread library calls would you make if you want your program to block and wait for a specific thread to finish executing?

   a) **`Pthread_block()`**

   b) **`Pthread_exit()`**

   c) **`Pthread_join()`**

   d) **`Pthread_wait()`**

   e) **`Pthread_yield()`**

   f) none of the above

Next Class

- User Space Threads
- Kernel Space Threads
- Threads in Linux

The End