University of Waterloo

SE350: Operating Systems

Mahmoud Salem

m4salem@uwaterloo.ca

Please email through Learn

Office hours will be decided if needed

Tutorial

- Chapter 4:
  - Review Questions: 4.1 – 4.7
  - Problem Set: 4.1, 4.2, 4.9

7th and 8th Editions have same numbering for those problems.

Chapter 4 Threads

Reference:

Operating Systems “Internals and Design Principles” 8th Edition – William Stallings

Characteristics embodied in concept of process

- Resource Ownership
  - process has virtual space to hold process image and the process is allocated ownership for main memory, I/O devices, ..
  - OS has to prevent unwanted interference between processes with respect to resources.
  - Typically relates to Task or Process
- Execution/Scheduling
  - OS schedules and dispatch processes based on their state (ready, ..) and priority.
  - Typically relates to Thread or lightweight process

Uses of Threads in a single-user multiprocessing system

- Foreground and Background work
  - Example: Spreadsheet program
    - Thread for menu display and user input
    - Thread for executing user command and process input
- Asynchronous processing
  - Thread for processing and Thread passes data to serial port RS232
- Speed of Execution
  - Multiple threads on multiple CPUs/Cores
- Modular program structure
  - easier to design and implement using threads.

Shared Resources by threads within a process

- All the threads of a process share the state and the resources of that process.
- They have the same address space and have access to same data
  - data altered by a thread, visible by others
- They have the same execution privileges for the resources accessed by the process
  - file opened for read by a thread, accessible by others

Thread Control Block Vs Process Control Block

(Review Question 3.1)

- Recall from Chp3 the elements for a process control block listed in Table 3.5 in your textbook and lecture slides 42-49.
- Now for a multithreaded OS, which of these elements should belong to a thread control block and which should belong to a process control block ?

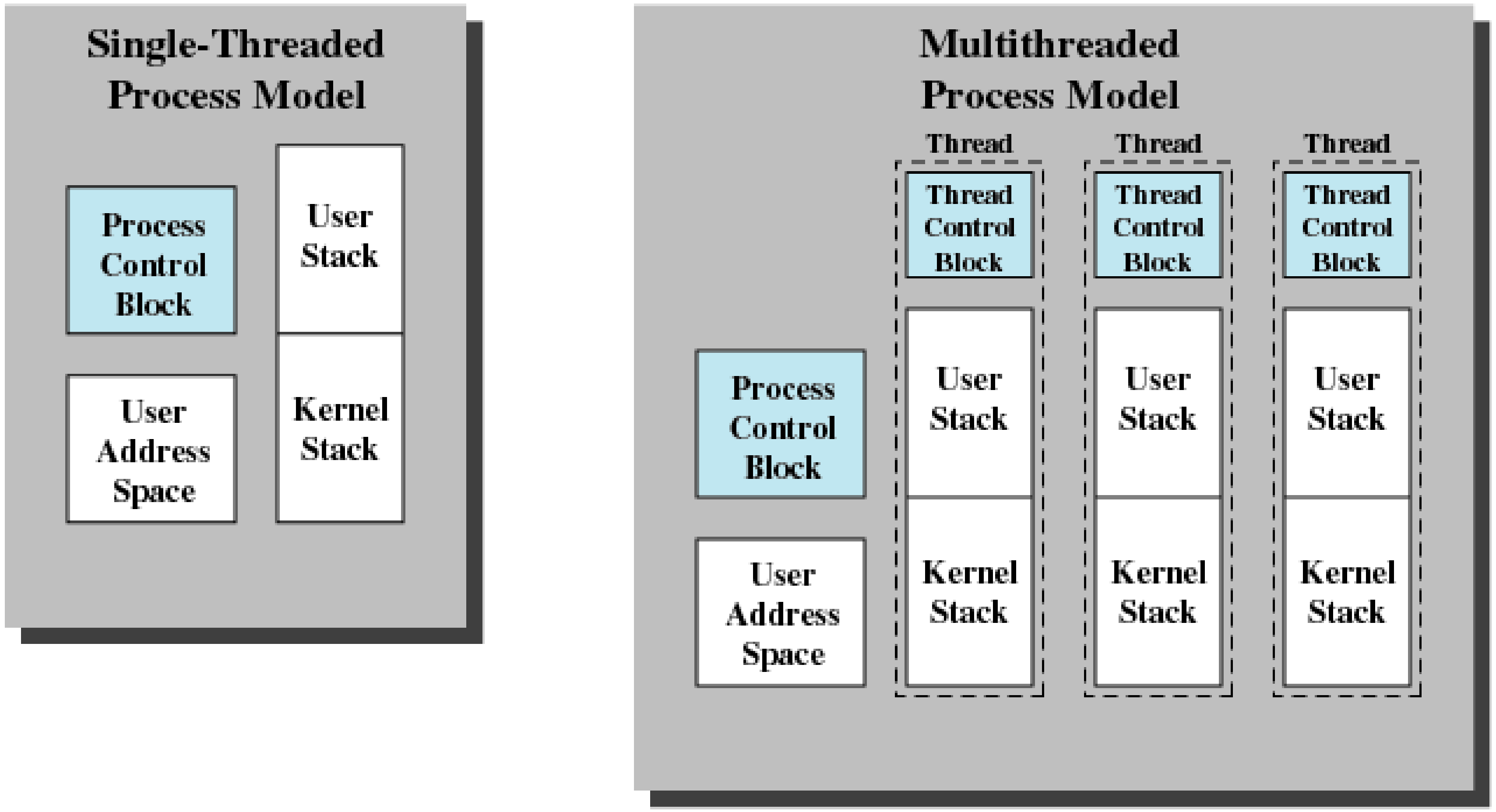


Figure 4.2 Single Threaded and Multithreaded Process Models

Table 3.5 Typical Elements of a Process Control Block

Process Identification

Identifiers
Numeric identifiers that may be stored with the process control block include

- Identifier of this process
- Identifier of the process that created this process (parent process)
- User identifier

Processor State Information

User-Visible Registers
A user-visible register is one that may be referenced by means of the machine language that the processor executes while in user mode. Typically, there are from 8 to 32 of these registers, although some RISC implementations have over 100.

Control and Status Registers
These are a variety of processor registers that are employed to control the operation of the processor. These include

- **Program counter:** Contains the address of the next instruction to be fetched
- **Condition codes:** Result of the most recent arithmetic or logical operation (e.g., sign, zero, carry, equal, overflow)
- **Status information:** Includes interrupt enabled/disabled flags, execution mode

Stack Pointers
Each process has one or more last-in-first-out (LIFO) system stacks associated with it. A stack is used to store parameters and calling addresses for procedure and system calls. The stack pointer points to the top of the stack.

Process Control Information

Scheduling and State Information

This is information that is needed by the operating system to perform its scheduling function. Typical items of information:

- **Process state:** Defines the readiness of the process to be scheduled for execution (e.g., running, ready, waiting, halted).
- **Priority:** One or more fields may be used to describe the scheduling priority of the process. In some systems, several values are required (e.g., default, current, highest-allowable).
- **Scheduling-related information:** This will depend on the scheduling algorithm used. Examples are the amount of time that the process has been waiting and the amount of time that the process executed the last time it was running.
- **Event:** Identity of event the process is awaiting before it can be resumed.

Data Structuring

A process may be linked to other process in a queue, ring, or some other structure. For example, all processes in a waiting state for a particular priority level may be linked in a queue. A process may exhibit a parent–child (creator–created) relationship with another process. The process control block may contain pointers to other processes to support these structures.

Interprocess Communication

Various flags, signals, and messages may be associated with communication between two independent processes. Some or all of this information may be maintained in the process control block.

Process Privileges

Processes are granted privileges in terms of the memory that may be accessed and the types of instructions that may be executed. In addition, privileges may apply to the use of system utilities and services.

Memory Management

This section may include pointers to segment and/or page tables that describe the virtual memory assigned to this process.

Resource Ownership and Utilization

Resources controlled by the process may be indicated, such as opened files. A history of utilization of the processor or other resources may also be included; this information may be needed by the scheduler.

Answer to Review Question 3.1

- Review Lecture Slides 6-7
- In general, resources and memory management are process-level
- Both processes and threads have identifier
- Process state info are related to process in general
- Typically the execution is on thread level, so the scheduling information are generally related to thread control block
- Both will require inter-communication

Review Question 3.2

- Mode switch between threads within a process is cheaper than mode switch between processes itself. Why ?
- Based on information in previous question, the state information to be saved for a thread switch is less than the information needed for a process switch.

Problem 4.1

- From advantages of using threads, easier thread creation within a process compared to a new process creation in addition to simpler communication between threads in a process.
- Regarding mode switch, Do you think mode switch between threads within the same process is easier/simpler than mode switch between threads of different processes ?

Problem 4.1 Solution

- Yes, because switching between threads of different processes will require switching between processes and this will involve more information to be saved as mentioned earlier.

Advantages for ULTs over KLTs

- Thread switch doesn’t require Kernel-mode privileges as thread management is handled by the user application (less switching overhead by saving 2 mode switches).
- Scheduling can be app-specific, without disturbing the OS scheduler.
  - round robin vs priority- based for different apps.
- ULT can run on any OS as thread library is a set of application functions.

Disadvantages of ULTs over KLTs

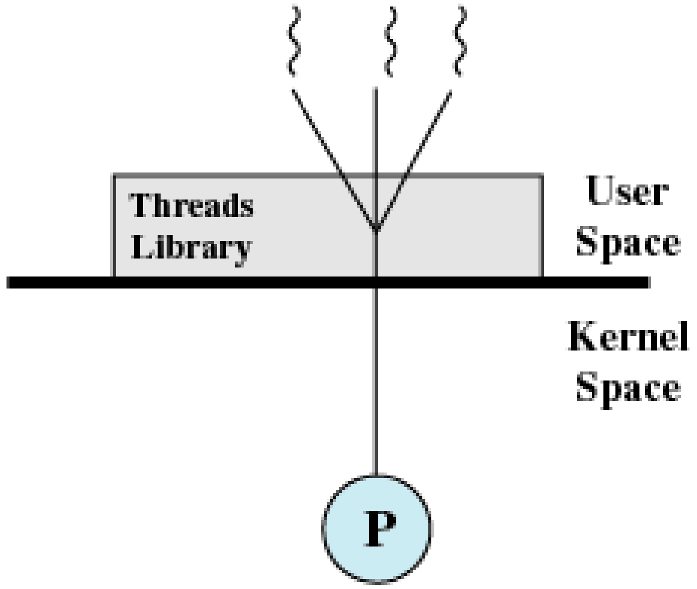
- Many system calls by the OS are blocking, all threads within the process are blocked.
- In pure ULT strategy, application can’t take advantage of multiprocessing.
  - Kernel assigns one process to one processor at a time, only a single thread within the process will be able to execute at a time.
  - Still interleaving cause a speedup, but some application code would benefit from simultaneous execution.

Problem 4.2

- In pure ULT strategy, why a system call by a thread will lead to blocking the whole threads for this process ?

Problem 4.2 Solution

- Thread structure of a process is not visible to the kernel (OS) where scheduling is only based on processes.



Jacketing

- Converts blocking system call to non-blocking system call.
- Uses application-level I/O jacket routine that has code to check if the I/O device is busy, If it is busy, the thread is blocked and another thread will gain control.
- When the blocked threads gain control again, the routine will check if the I/O device is ready again.

Problem 4.9

- The code in file thread2.c which can be found here has main() which creates another thread with id “mythread” and the function is “thread_function”
- We have two threads, the main program and the thread function
- The main thread waits for the other to complete before it terminates

What does each thread do ? What do you expect to happen ?

- Look into code !
- Main program creates a thread
- Then the code in each thread increments the global variable “*myGlobal*” 20 times.
- Typically, having 2 programs incrementing a global variable 20 times, should result in incrementing it 40 times.

What will actually happen ?

- *myGlobal* will equal to 21 at the end of the execution – why !!
- Back to Code
- *sleep()* function causes a thread to be suspended for an amount of time.
- The issue in the local copy within the *thread_function()*. The increment effect of main() is overwritten !! once we are back to the *thread_function()* after suspension.

More clarification !

- Effect of 20 increments of the *thread_function()*
- Effect of only 1 increment by the *main()* thread is counted either before the 20 iterations or after the 20 iterations of the *thread_function()* results into 21 increments total.
- The other 19 increments of main thread will be overwritten.
- Only assuming roughly same sleep time.
- Try it on your machine, change the sequence and observe the results !

Questions ?