Programs, Processes, and Threads

- Programs, Processes, and Threads (Chapter 2)
- Processes in UNIX (Chapter 3)
Processes Management

• What is a process?
• How to control processes.
• How to allocate the available resources to the execution of the processes (scheduling)
• How to coordinate processes among themselves (synchronization)

Processes and Process Control

• Q: What is a process?
• Process as execution of a Program
• We can trace the execution of a process
• Process as minimal entity for resource allocation (for example memory).
Simple Memory Layout of a Running Program

The Execution Trace of Processes

- Two processes and a dispatcher

<table>
<thead>
<tr>
<th>Traces of processes A and B</th>
<th>α</th>
<th>β</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>α+1</td>
<td>β+1</td>
</tr>
<tr>
<td>α+2</td>
<td>β+2</td>
<td></td>
</tr>
<tr>
<td>α+3</td>
<td>β+3</td>
<td></td>
</tr>
<tr>
<td>α+4</td>
<td>β+4</td>
<td></td>
</tr>
<tr>
<td>α+5</td>
<td>β+5</td>
<td></td>
</tr>
<tr>
<td>α+6</td>
<td>β+6</td>
<td></td>
</tr>
<tr>
<td>α+7</td>
<td>β+7</td>
<td></td>
</tr>
<tr>
<td>α+8</td>
<td>β+8</td>
<td></td>
</tr>
<tr>
<td>α+9</td>
<td>β+9</td>
<td></td>
</tr>
<tr>
<td>α+10</td>
<td>β+10</td>
<td></td>
</tr>
<tr>
<td>α+11</td>
<td>β+11</td>
<td></td>
</tr>
</tbody>
</table>

- Trace of dispatcher

<table>
<thead>
<tr>
<th>δ</th>
<th>δ+1</th>
<th>δ+2</th>
<th>δ+3</th>
<th>δ+4</th>
</tr>
</thead>
<tbody>
<tr>
<td>δ+5</td>
<td>δ+6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>δ+7</td>
<td>δ+8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>δ+9</td>
<td>δ+10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| δ+11                        | δ+12    |...
States of a Process

- **User view:** A process is executing continuously
- **In reality:** Several processes compete for the CPU and other resources
- A process may be
  - **running:** it holds the CPU and is executing instructions
  - **blocked:** it is waiting for some I/O event to occur
  - **ready:** it is waiting to get back on the CPU

Process Creation

- Submission of a batch job
- User logs on
- Create process to provide service such as printing
- Spawned by existing processes

- In UNIX:
  all processes created by `fork()` system call
**Example: Vanilla Command Interpreter**

```c
char command[MAX_COMMAND_LENGTH];
do {
    command = read_command(stdin);
    if (fork() != 0) {
        /* parent */
        if (last_char(command) != ';') {
            /* run in foreground, i.e. wait */
            waitpid(-1, &status, ...);
        }
    } else {
        /* child */
        execve(command, ...);
    }
} while (strcmp(command, "exit") != 0); /* ?!? */
```

**Suspended Processes**

- **start**
- **suspended ready**
- **ready**
- **running**
- **suspended blocked**
- **blocked**

The process state transition diagram shows the typical states a process can be in: start, suspended ready, ready, running, suspended blocked, and blocked. The diagram illustrates how processes can transition between these states under various conditions.
The Process Control Block (PCB)

- Mechanism of a process switch:

  - Preempt Process A and store all relevant information.
  - Load information about Process B and continue execution.
  - Preempt Process B and store all relevant information.
  - Load information about Process A and continue execution.

- The PCB contains all information specific to a process.

Example for the Use of PCBs: Process Queues
Elements of a PCB

| process identification | process id  
|                        | parent process id  
|                        | user id  
|                        | etc...  

| processor state information | register set  
|                            | condition codes  
|                            | processor status  

| process control information | process state  
|                            | scheduling information  
|                            | event (wait-for)  
|                            | memory-mgmt information  
|                            | owned resources  

Processes in UNIX

fork() → created → ready swapped

- not enough memory → swap in
- enough memory → ready
- swap out → swapped
- wake up → sleep
- swap out → sleep swapped
- preempt → preemption
- reschedule process → kernel running
- system call → interrupt
- return to user

sleep in memory → sleep

user running → exit

zombie
Programs, Processes, and Threads

- Programs, Processes, and Threads (Chapter 2)
- Processes in UNIX (Chapter 3)

Threads

- Traditionally, processes interact very little:
  - Processes as jobs
    - in batch queue
  - User processes
  - Kernel

- This is not true in modern systems: Some applications may want to have multiple, tightly-coupled processes.
CPSC 313: Intro to Computer Systems

Problems with traditional (heavy-weight) processes

- Heavy-weight processes have separate address spaces:
  - Process creation is expensive
  - Process switch is expensive
  - Sharing memory areas among processes non-trivial

Threads

- Threads share address space:
  - Thread creation much simpler than process creation (no need to create and initialize address space, etc.)
  - Thread switch simple
  - Threads fully share the address space

- Convenience
  - Communication between threads
- Efficiency
  - Multiprogramming within a process (Netscape vs. Mosaic)
  - Multiprocessors
User-Level vs. Kernel-Level Threads

- **User-level**: kernel not aware of threads
- **Kernel-level**: all thread-management done in kernel

Potential Problems with Threads

- **General**: Several threads run in the same address space:
  - Protection must be explicitly programmed (by appropriate thread synchronization)
  - Effects of misbehaving threads limited to task
- **User-level threads**: Some problems at the interface to the kernel: With a single-threaded kernel, as system call blocks the entire task.
CPSC 313: Intro to Computer Systems

Singlethreaded vs. Multithreaded Kernel

- Protection of kernel data structures is trivial, since only one process is allowed to be in the kernel at any time.
- Special protection mechanism is needed for shared data structures in kernel.

Threads in Solaris 2.x

- Processes
- User-level threads
- Light-weight processes
- Kernel threads
- Kernel
- CPUs