Processes and Threads

Learning Outcomes

• An understanding of fundamental concepts of processes and threads
  • I’ll cover implementation in a later lecture

Major Requirements of an Operating System

• Interleave the execution of several processes to maximize processor utilization while providing reasonable response time
• Allocate resources to processes
• Support interprocess communication and user creation and management of processes

Processes and Threads

• Processes:
  • Also called a task or job
  • Memory image of an individual program
  • “Owner” of resources allocated for program execution
  • Encompasses one or more threads
• Threads:
  • Unit of execution
  • Can be traced
  • list the sequence of instructions that execute
  • Belongs to a process
  • Executes within it.

Execution snapshot of three single-threaded processes (No Virtual Memory)

Logical Execution Trace

5000  8000  12000
5001  8001  12001
5002  8002  12002
5003  8003  12003
5004  12004
5005  12005
5006  12006
5007  12007
5008  12008
5009  12009
5010  12010
5011  12011

Figure 3.1 Snapshot of Example Execution (Figure at Instruction Cycle 13)

Figure 3.2 Traces of Processes of Figure 3.1
Combined Traces
(Actual CPU Instructions)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Time (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
</tr>
</tbody>
</table>

What are the shaded sections?

1. **Multiprogramming of four programs**
2. **Conceptual model of 4 independent, sequential processes (with a single thread each)**
3. **Only one program active at any instant**

---

**Summary: The Process Model**

- Process and thread models of selected OSes
  - Single process, single thread
    - MSDOS, simple embedded system
  - Single process, multiple threads
    - OS/161 as distributed
  - Multiple processes, single thread
    - Traditional UNIX
  - Multiple processes, multiple threads
    - Modern Unix (Linux, Solaris), Windows

Note: Literature (incl. Textbooks) often do not cleanly distinguish between processes and threads (for historical reasons)

---

**Process Creation**

Principal events that cause process creation

1. System initialization
   - Foreground processes (interactive programs)
   - Background processes
     - Email server, web server, print server, etc.
     - Called a daemon (unix) or service (Windows)
2. Execution of a process creation system call by a running process
   - New login shell for an incoming ssh connection
3. User request to create a new process
4. Initiation of a batch job

Note: Technically, all these cases use the same system mechanism to create new processes.

---

**Process Termination**

Conditions which terminate processes

1. Normal exit (voluntary)
2. Error exit (voluntary)
3. Fatal error (involuntary)
4. Killed by another process (involuntary)
Implementation of Processes

- A processes’ information is stored in a process control block (PCB)
- The PCBs form a process table
  - Reality can be more complex (hashing, chaining, allocation bitmaps,...)

Example fields of a process table entry

Process/Thread States

1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Some Transition Causing Events

Running → Ready
- Voluntary Yield()
- End of timeslice

Running → Blocked
- Waiting for input
  - File, network
  - Waiting for a timer (alarm signal)
  - Waiting for a resource to become available

Scheduler

- Sometimes also called the dispatcher
  - The literature is also a little inconsistent on with terminology.
- Has to choose a Ready process to run
  - How?
    - It is inefficient to search through all processes

The Ready Queue

Enter → Queue → Dispatch → Process → Exit

(b) Queueing diagram
What about blocked processes?

• When an unblocking event occurs, we also wish to avoid scanning all processes to select one to make Ready

Using Two Queues

Threads

The Thread Model

(a) Three processes each with one thread
(b) One process with three threads

The Thread Model – Separating execution from the environment.

Per process items
Address space
Global variables
Open files
Child processes
Pending alarms
Signals and signal handlers
Accounting information

Per thread items
Program counter
Registers
Stack
State

• Items shared by all threads in a process
• Items private to each thread

Threads Analogy

The Hamburger Restaurant
Observation: Computation State

- **Thread Model**: State implicitly stored on the stack (local variables in the function).
- **Finite State (Event) Model**: State explicitly managed by program.

The Thread Model

Each thread has its own stack.
Thread Model

- Local variables are per thread
  - Allocated on the stack
- Global variables are shared between all threads
  - Allocated in data section
  - Concurrency control is an issue
- Dynamically allocated memory (malloc) can be global or local
  - Program defined (the pointer can be global or local)

Thread Usage

A word processor with three threads

Thread Usage

Rough outline of code for previous slide
(a) Dispatcher thread
(b) Worker thread – can overlap disk I/O with execution of other threads

Thread Usage

Three ways to construct a server

<table>
<thead>
<tr>
<th>Model</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threads</td>
<td>Parallelism, blocking system calls</td>
</tr>
<tr>
<td>Single-threaded process</td>
<td>No parallelism, blocking system calls</td>
</tr>
<tr>
<td>Finite-state machine</td>
<td>Parallelism, nonblocking system calls, interrupts</td>
</tr>
</tbody>
</table>

Summarising “Why Threads?”

- Simpler to program than a state machine
- Less resources are associated with them than multiple complete processes
  - Cheaper to create and destroy
  - Shares resources (especially memory) between them
- Performance: Threads waiting for I/O can be overlapped with computing threads
  - Note if all threads are compute bound, then there is no performance improvement (on a uniprocessor)
- Threads can take advantage of the parallelism available on machines with more than one CPU (multiprocessor)