Processes vs. Threads

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Today’s Topics

- What is the process?
- What is the thread?
Process

What is the process?

- An instance of a program in execution.
- An encapsulation of the flow of control in a program.
- A dynamic and active entity.
- The basic unit of execution and scheduling.
- A process is named using its process ID (PID).
- Job, task, or sequential process
- A process includes:
  - CPU contexts (registers)
  - OS resources (memory, open files, etc.)
  - Other information (PID, state, owner, etc.)
Process Address Space

PC → Code

Data

Heap

Shared Data

SP → Stack
Using Processes

#include <sys/types.h>
#include <unistd.h>

int main()
{
    int pid;

    if ((pid = fork()) == 0) /* child */
        printf ("Child of %d is %d\n", getppid(), getpid());
    else /* parent */
        printf ("I am %d. My child is %d\n", getpid(), pid);
}
% ./a.out
I am 31098. My child is 31099.
Child of 31098 is 31099.

% ./a.out
Child of 31100 is 31101.
I am 31100. My child is 31101.
What is a Thread?

- **A thread of control (or a thread)**
  - A sequence of instructions being executed in a program.
  - Usually consists of
    - a program counter (PC)
    - a stack to keep track of local variables and return addresses
    - registers
  - Threads share the process instructions and most of its data.
    - A change in shared data by one thread can be seen by the other threads in the process
  - Threads also share most of the OS state of a process.
Using Threads

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

void *hello (void *arg)
{
    printf ("hello, world\n");
    ...
}

int main()
{
    pthread_t tid;

    pthread_create (&tid, NULL, hello, NULL);
    printf ("hello from main thread\n");
    ...
}
```
Address Space with Threads

- Code
- Data
- Heap
- Stack

Program

code
data

PC (T2) →
PC (T1) →
PC (T3) →

SP (T2) →
SP (T3) →
SP (T1) →
Processes vs. Threads

- A thread is bound to a single process.
- A process, however, can have multiple threads.
- Sharing data between threads is cheap: all see the same address space.
- Threads become the unit of scheduling.
- Processes are now containers in which threads execute.
- Processes become static, threads are the dynamic entities.
## Classification

<table>
<thead>
<tr>
<th># threads per addr space:</th>
<th># of addr spaces:</th>
<th>One</th>
<th>Many</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MS/DOS Early Macintosh</td>
<td>Traditional UNIX</td>
<td></td>
</tr>
<tr>
<td>One</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Many Oses (VxWorks, uClinux, ..)</td>
<td>Pintos</td>
<td>Mach, OS/2, Linux, Windows, Mac OS X, Solaris, HP-UX</td>
<td></td>
</tr>
</tbody>
</table>