CS 350 Notes

Matthew Visser

Oct 6, 2011

1 Processes

The kernel keeps track of who made a process, resources it holds, address space, *etc.* This is stored in a struct called the *process control block*.

`struct thread` contains a lot of information about a running process as well, but the subprocess that is a thread.

A process with one thread is called a *sequential process*. A process with more than one thread is called a *concurrent process*.

1.1 Timesharing

Context switching happens inside the kernel. When a thread enters the kernel, this is an opportunity to do a context switch.

The role of the timer interrupt is to make sure that a thread periodically enters the kernel. Pre-emption can still happen any time a thread enters the kernel.

1.2 System Calls for Process Management

See slide 35 of `proc.pdf`.

`fork` creates a clone of the calling process. Fork returns from both the calling process, *i.e.*, the original one, and the child process, *i.e.*, the new one. When

it returns, if it is in the child process, it returns 0, if it is the parent, it returns the process ID of the new process.

`execv` replaces the address space of the calling process with an address space loaded from an executable file.

`waitpid` allows a process to wait for another process to finish.

`getpid` gets the process id

`getusage` tells you how many resources you are using and how much.

There is a distinction between *kernel threads* and *user threads*. Kernel threads are dispatched by the kernel. User threads are possibly not. User threads do not get pre-empted because they are in user-space, so they need to yield to each other. The reason for having these user threads is because kernel threads are fairly heavy-weight and user threads are lighter. Sometimes, operating systems don't support multiple threads per user process as well. There are some threads used by the kernel for kernel "housekeeping" only.

2 Memory

2.1 Virtual vs Physical Addresses

Memory has both physical and virtual addresses.

- physical addresses
  - Physical addresses are provided by the machine. There is only one physical address per machine.
  - The size of the physical address determines the amount of addressable memory.
  - MIPS has 32-bits, so we can only have 2^{32}bytes = 4GB of physical memory.
- Virtual addresses
  - There is one virtual address space *per process*.

- Mapping between physical and virtual addresses are managed by the operating system.

Advantages for virtual memory are:

- Convenience — all processes start at address 0.
- Easier to do paging.
- Protection and isolation of processes — the kernel can make sure that processes can't access other processes memory, or the kernel's memory.
- Flexibility for the kernel to manage memory.

Virtual addresses are 32 bits, just like physical addresses.

What's in virtual address space?

- text — program code and read-only memory.
- data — heap memory.
- stack — starts at an address (`0x7fffffff`) and grows down.

2.2 Address Translation

- Hardware provides a *memory management unit (MMU)* which includes a *relocation register*.
- In *dynamic relocation*, the processes are loaded in a continuous block of memory. The OS puts the start address of the memory in the relocation register.
- The *limit register* makes sure that the virtual address doesn't go over the amount of memory allocated for the process.
- There is more than one address translation per instruction. One for translating the address of the instruction, and any more for any addresses the instruction needs. This means that the OS can't be too involved, and this is why we have the MMU, which is fast because it's hardware.
- *External fragmentation* can happen in dynamic relocation. If we don't have enough continuous space in physical memory, then we have a problem. There are heuristics for avoiding this. These are:
  - First fit — take the first section that fits

- Best fit — take the smallest continuous section possible
- Worst fit – take the largest continuous section possible

¡++¿

¡++¿