Operating Systems

Tutorial 2 & 16

Michael Tänzer

os-tut@nhng.de
http://os-tut.nhng.de

Calendar Week 46

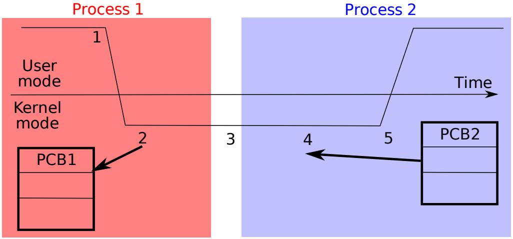
Outline

1. Review
2. Processes
   - Context Switch
   - PCB
   - Short-, Medium- and Long-Term Scheduler
   - CPU- and I/O-Bound Processes
3. Processes in UNIX
   - `fork`
   - `execve`
4. Race Conditions and Deadlocks
   - Race Condition
   - Deadlock

True or False

- Reading from a user provided buffer is always save.
- You could implement syscalls using exceptions instead of a dedicated `trap` instruction.
- If the virtual machine is as complex but very different from the machine it is running on the VM monitor will be large and there will be a high performance impact.

Describe how the kernel performs a context switch



1. Interrupt/exception/syscall occurs
2. Save execution state of the interrupted/preempted process/thread to its PCB
3. Maybe switch address spaces, flush caches etc.
4. Load execution state of process/thread to be run next from its PCB
5. Return to user mode

What does a process control block (PCB) contain?

- Process state
- Process ID (PID)
- Program counter
- Registers (including stack pointer)
- Credentials (UID, GID)
- Accounting, scheduling, memory management and I/O status information
- Most entries are only valid if the process is currently not running

Difference between short-, medium- and long-term scheduler

- Short-term scheduler
  - Which process should run next?
  - That's the one we will focus on
- Medium-term scheduler
  - Which processes should be swapped out to disk?
  - Influence degree of multiprogramming
- Long-term scheduler
  - What processes should be loaded?
  - Only in batch systems

CPU- vs. I/O-bound processes

- CPU-bound processes rarely do I/O $\Rightarrow$ they are unlikely to block
- I/O-bound only perform short CPU bursts between doing I/O $\Rightarrow$ they are likely to block very often

Why is a good mixture of CPU- and I/O-bound processes preferable?

- Ensure good utilisation of ressouces (CPU and I/O devices)

What does the `fork` syscall do?

- Create a child process which is identical to the parent process
  - Same memory contents
  - Same open files
  - Same register contents (including program counter)
  - ...
- Exceptions:
  - Different process ID (PID)
  - Different address space (contents are the same though)
  - ...
- Returns child's PID to the parent and 0 to the child

Example

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
int main(int argc, char **argv){
    /* Parent prints "I am the parent!\n",
     *  "Child PID is %d\n", "Child terminated\n"
     *
     * Child prints "I am the child!\n"
     *
     * waitpid(pid, NULL, 0) will wait for the
     * child with the specified PID to exit
     */
    return 0;
}
```

Example

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
int main(int argc, char **argv){
    printf("I am the parent!\n");
    pid_t pid = fork();

    if (pid > 0){
        printf("Child PID is %d\n", pid);
        waitpid(pid, NULL, 0);
        printf("Child terminated\n");
    }
    else if (pid == 0){
        printf("I am the child!\n");
    }
    else{
        printf("Error. Fork failed\n");
    }

    return 0;
}
```

Is `fork` sufficient to implement a shell?

- `fork` allows us to create a new process
- We need a way to replace the code of the current process with the one that shall be executed
- `execve` does that
- With `execve` the shell can
  1. Create a new process (`fork`)
  2. Child: Possibly make some adjustments (e. g. redirect standard output)
  3. Child: Replace the code of the shell with the code of the programme to be run (`execve`)
  4. Parent: Wait for the programme to terminate (`wait`)
  5. Show prompt

What is a race condition?

A race condition is a situation where the correctness of the result of

- Multiple operations
- Executed by multiple activities (threads or processes)
- Depends on the scheduling order of those activities

How to avoid race conditions

- Make sure critical sections are executed atomically
- Usually done using synchronisation primitives
  - Locks
  - Semaphores
  - Monitors

What is a deadlock?

A deadlock is a situation where

- A set of activities can't make any progress
- because each activity in the set is waiting for an event
- this event can only be emitted by another activity in the set

How to avoid deadlocks

- One possibility is to always access ressources in the same (global) order

Questions & Comments

Any questions or comments?

The End

The End