Operating Systems

Tutorial 2 & 16

Michael Tänzer

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

Calendar Week 47

Outline

True or False

- The PID, process state and program counter entries in the PCB are valid when the process is currently running.
- After a `fork` changes to variables will be visible to both child and parent.
- When accessing synchronised resources in the same order no deadlock can occur.
- When protecting a variable with a lock no other activity can access it without a (digital) key.

ls | less

```c
#include <stdio.h>
#include <unistd.h>
#define READ_END    0
#define WRITE_END   1
int main( int argc, char **argv ){
    int pid;
    int pipefd[2];
    pipe( pipefd );
    pid = fork();
    if ( pid == 0 ){
        dup2( pipefd[WRITE_END], STDOUT_FILENO );
        close( pipefd[READ_END] );
        execlp( "ls", "ls", NULL );
    }
    else if ( pid > 0 ){
        pid = fork();
        if ( pid == 0 ){
            dup2( pipefd[READ_END], STDIN_FILENO);
            close( pipefd[WRITE_END] );
            execlp( "less", "less", NULL );
        }
        else{
            close( pipefd[READ_END] );
            close( pipefd[WRITE_END] );
            wait(NULL);
            wait(NULL);
        }
    }
}
```

Compare `CreateProcess` with `fork`

Strengths and Weaknesses

- `CreateProcess` ≈ `fork` + `execve`
- \+ Saves one system call
- – Less flexible (can't use processes like some kind of pseudo-threads)
- – Modifying the context of the new process is more complex
  - In UNIX you just `fork` change the context and then do the `execve` (e. g. redirect `stdin/stdout` as shown in the previous example)
  - In Windows you have to build a `STARTUPINFO` structure and pass a pointer to it to `CreateProcess`

Where might it be useful to use threads?

- Video games: One thread for graphics, physics, AI, ...
- Web server: One thread listens for incoming connections and dispatches them each to a new thread
- Image processing: Partition the image into *n* parts and process each part in a separate thread

Multi-threading vs. multiple processes

+ Lower overhead
  - Thread creation
  - Switching between threads of the same process (no address space switch needed)
+ Data sharing: Threads can easily work on the same data and communicate via shared memory (most operating systems also allow sharing of memory between processes but that's more complicated)
– Less isolation: Need to be careful when determining which variables should be synchronised

Discuss the three threading models

- Many-to-One
- One-to-One
- Many-to-Many

With regard to

- Performance
- Behaviour on blocking syscalls
- Utilisation of multi processor systems
- Kernel awareness

Many-to-One

- Fast thread switch and creation (kernel not involved)
- A blocking syscall in one thread will block the whole process
- Can only use one processor
- Kernel does not know about threads (i. e. will also work on kernels which don't have a thread implementation)

One-to-One

- Each thread creation and switch requires kernel invocation
- A blocking syscall will only affect the thread doing the syscall
- Can use as many processors as there are threads
- Kernel does all the bookkeeping and management of threads $\Rightarrow$ the kernel needs to have a thread implementation

Many-to-Many

- Creation and switch of kernel level threads needs kernel invocation but on user level threads it doesn't
- A blocking syscall in one thread will block those threads that are mapped to the same kernel level thread
- Can use as many processors as there are threads
- Kernel needs to know about threads. Also the two schedulers will want to exchange data to make informed decisions (even more complex)

What happens when a multi-threaded process in Linux calls `fork`

- Only the thread calling `fork()` will be duplicated

$\Rightarrow$ child processes created with `fork` are always single-threaded

What is the purpose of scheduling?

- Find a mapping: processes $\longrightarrow$ resources so that each process will eventually get the resources it needs
- Try to maximise some quality metrics (goals in policyspeak) e. g. resource utilisation
- We'll focus on CPU scheduling

What metrics can be used to estimate the quality of a scheduling policy?

What criteria should be maximised which minimized?

Utilization	percentage of time a resource is not idle
Troughput	number of requests (in CPU scheduling processes/threads) completed per unit of time
Turnaround time	time from submission of a request to its completion
Response time	time from submission of a request until the first response is produced
Waiting time	time a request is not being processed (in CPU scheduling time spent in the ready queue)

Example

Given:

- Three batch processes (which never do blocking syscalls)
  - P_1: execution time $T_1^e = 7$, arrives at $T_1^a = 2$
  - P_2: execution time $T_2^e = 3$, arrives at $T_2^a = 0$
  - P_3: execution time $T_3^e = 1$, arrives at $T_3^a = 7$
- FIFO scheduling

Task:

- Draw Gantt chart
- Calculate average waiting time T^w
- Calculate average turnaround time T^t

Questions & Comments

Any questions or comments?

The End

The End