Concurrent Programming

Dr. Bystrov

School of Engineering

Newcastle University

Aims

- Process
  - Concept
  - Modelling
  - Examples

Concurrent execution – baseline

The baseline idea is very simple:

- CPU switches fast between the concurrent activities (tasks, threads or processes)
- mechanisms and libraries for task control
- means for data communication
- means of protecting the shared data from access conflicts

The details are not simple and need to be discussed...

Process

- Process as an instance of the executed program.
- foreground – one process or pipeline at a time:
  - `echo “proc”`
  - `sleep 5; echo proc1; sleep 10; echo proc2`
  - `cat | tr a-z A-Z`
- background – many processes concurrently:
  - `sleep 5 &`
  - `(sleep 5; echo proc1; sleep 10; echo proc2) &`
  - `(sleep 1; echo proc1) & echo proc2 &`

Fork-join example

```
# Define functions
process_model()
 {
  xterm -e "echo $1 ; echo press Ctl-c to finish ; sleep 1h"
  return 0
 }

# Start process_1 first
process_model process_1

# Once it has finished start
# process_2 and process_3 concurrently
process_model process_2 &
pid_process_2=$!
process_model process_3

# After both processes are finished start process_4
wait $pid_process_2
process_model process_4
```

Processes in C

fork – creates a new child process, the exact copy of the parent process;

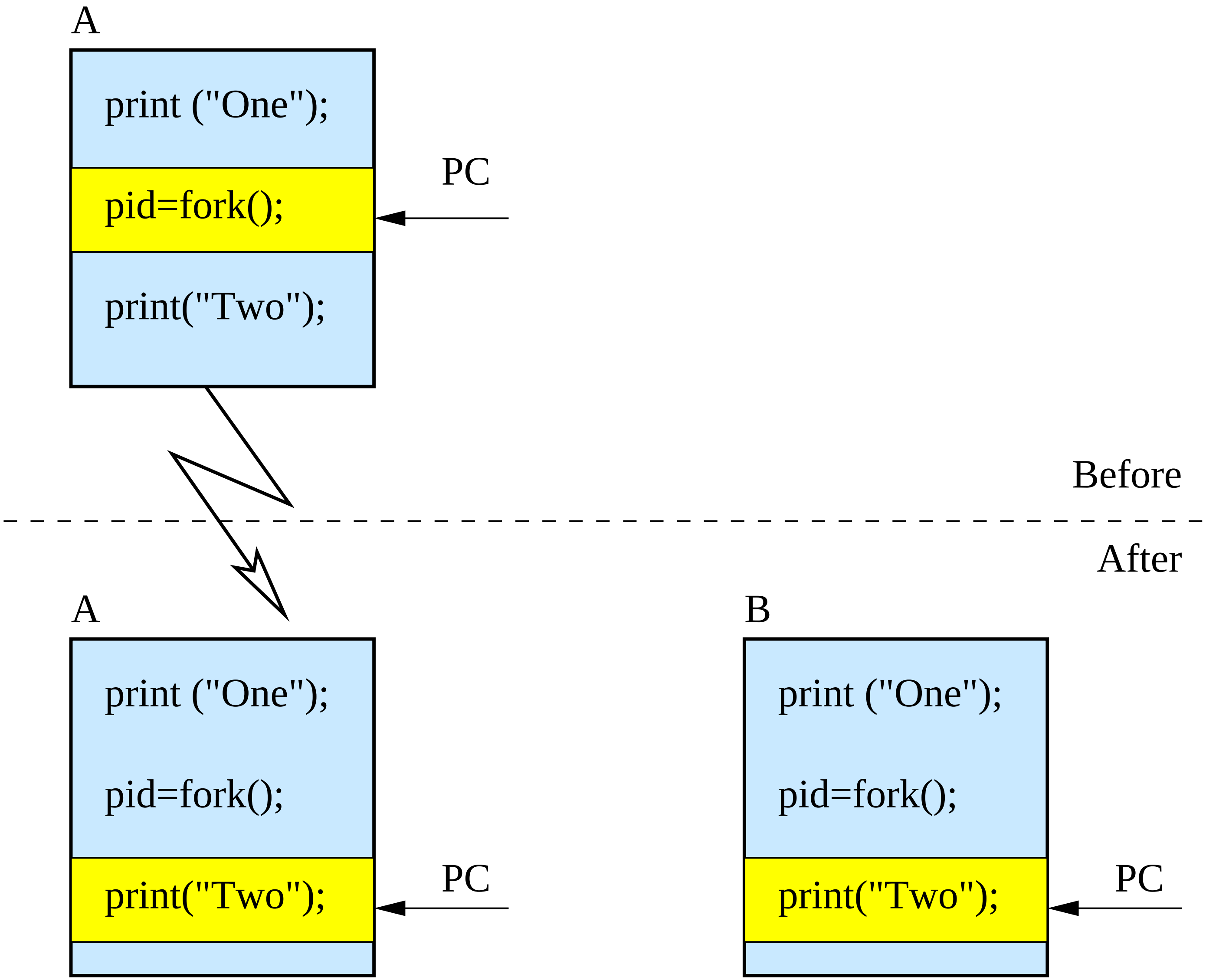
exec – replaces the task of the process by overwriting it;

wait – primitive synchronisation by waiting for the process to finish;

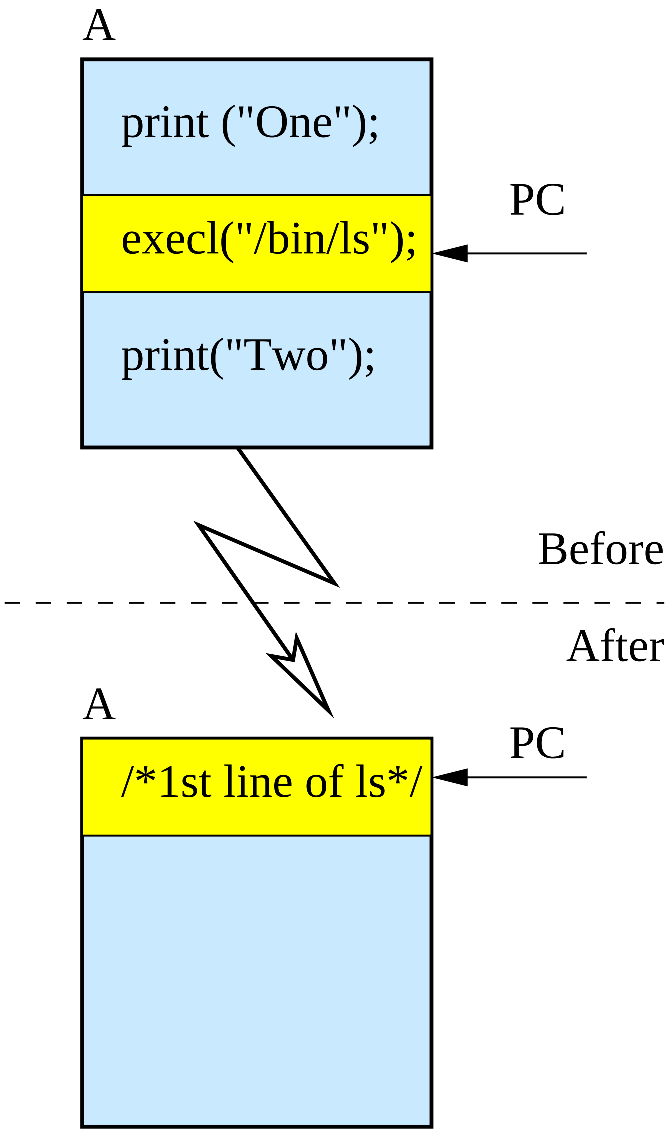
exit – stops the process.

fork + exec – creates a new process, different from the parent.

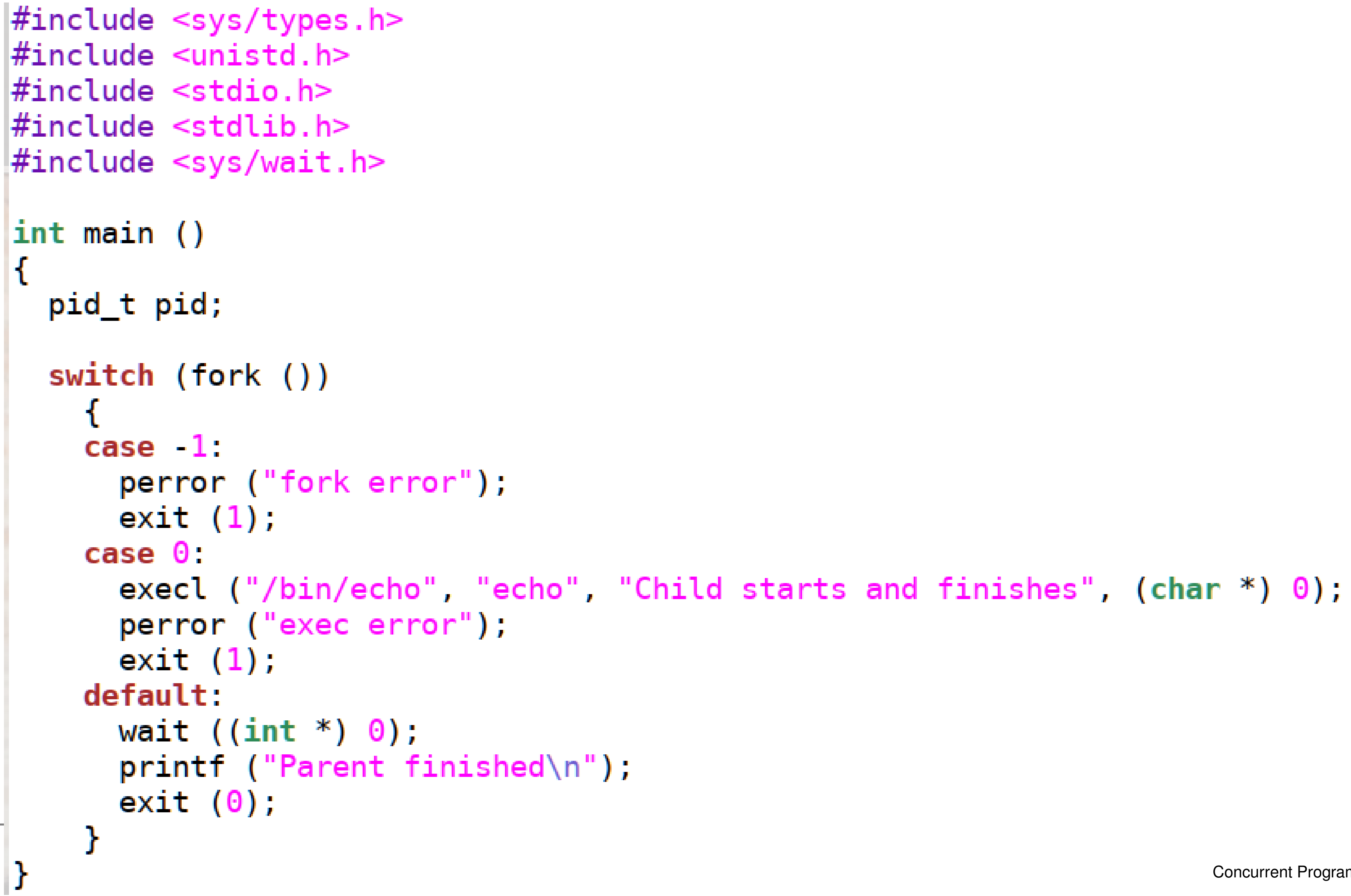
fork()


A
print ("One");
pid=fork();
print("Two");
PC
Before
After
A
print ("One");
pid=fork();
print("Two");
PC
B
print ("One");
pid=fork();
print("Two");
PC

Exec


A
print ("One");
execl("/bin/ls");
print("Two");
PC
Before
After
A
/*1st line of ls*/
PC

Example



```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>

int main ()
{
  pid_t pid;

  switch (fork ())
    {
    case -1:
      perror ("fork error");
      exit (1);
    case 0:
      execl ("/bin/echo", "echo", "Child starts and finishes", (char *) 0);
      perror ("exec error");
      exit (1);
    default:
      wait ((int *) 0);
      printf ("Parent finished\n");
      exit (0);
    }
}
```

Conclusions

- The schedulers implement the mechanism of concurrency
- Fork-join, choice-merge, arbitration when accessing common resource
- Two types of dependency between processes
  - imposed by start-termination control
  - imposed by data communication
- Modelling concurrent computations with Petri nets
- Experiments