Computer Labs: Processes

2º MIEIC

Pedro F. Souto (pfs@fe.up.pt)

October 16, 2015

Minix 3 Notes: `driver_receive()` is not Polling

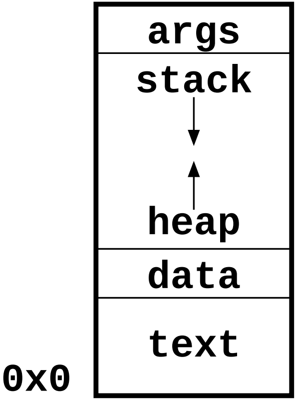
```
while( 1 ) { /* You may want to use a different condition
    /* Get a request message. */
    if ( driver_receive(ANY, &msg, &ipc_status) != 0 ) {
        printf("driver_receive failed with: %d", r);
        continue;
    }
    if (is_ipc_notify(ipc_status)) { /* received notificat
        switch (_ENDPOINT_P(msg.m_source)) {
        case HARDWARE: /* hardware interrupt notification
            if (msg.NOTIFY_ARG & irq_set) { /* subscribed
                ... /* process it */
            }
            break;
        default:
            break; /* no other notifications expected: do
        }
    } else { /* received a standard message, not a notific
```

`driver_receive()` what if the process' "IPC queue" is empty?

(Sequential) Process

Abstracts a running program

```
int main(int argc, char *argv[], char* envp[])}
```



- `args` Command line args and environment variables de ambiente.
- `stack` *Activation frames/records* corresponding to function calls
- `heap` Dynamically allocated memory (e.g using `malloc`)
- `data` Memory allocated statically (by the compiler) (e.g. the "`Hello, World!`" string)
- `text` Machine instructions

Minix is a multiprogramming OS

```
$ ps ax | more
  PID TTY   TIME CMD
 (-4)    ?  0:46 idle
 (-3)    ?  0:00 clock
 (-2)    ?  0:00 system
 (-1)    ?  0:00 kernel
    5    ?  0:00 pm
    7    ?  0:01 vfs
    4    ?  0:00 rs
    8    ?  0:00 memory
    9    ?  0:00 log
   10    ?  0:00 tty
    3    ?  0:00 ds
   12    ?  0:00 vm
   13    ?  0:00 pfs
    6    ?  0:00 sched
    1    ?  0:00 init
   -- more (43 in all)
```

- And so are Linux and all Windows OSs since XP (at least)

OS support multiple processes (multiprogramming)
for reasons of **efficiency**

Multiprogramming and Efficiency

Problem Processes need to access to I/O devices (monitor, keyboard, mouse, disk, network ...)

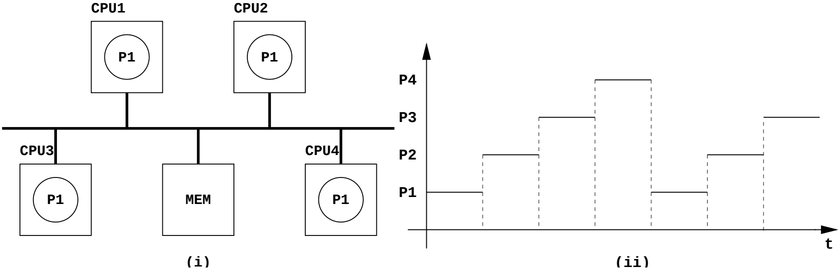
Parameter	Time
CPU cycle	1 ns (1 GHz)
Cache access	~ 2ns
Memory access	~ 10 ns
Disk access	~10 ms

Solution while a process waits for an I/O operation to complete, the OS can allocate the processor to another process:

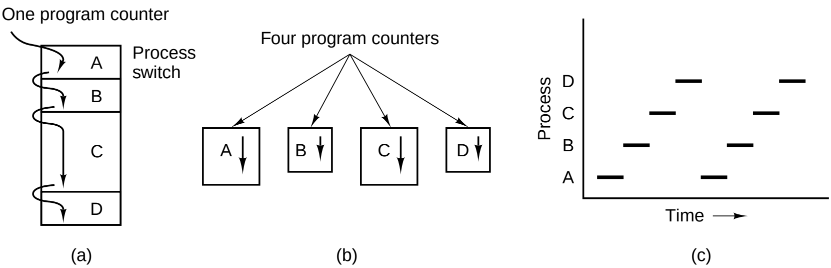
- Upon completion of the I/O operation, the I/O device can generate an interrupt

Multi-process Execution (1/2)

- In a multiprocessor/multicore system (i), each processor/core can execute a different process
- In a uniprocessor system (ii), the OS allocates the processor to the diferent processes (the processor is a resource shared by the different processes): *pseudo-parallelism*.



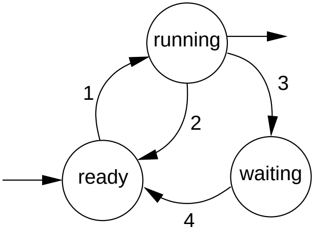
Multi-process Execution (2/2)



- The processor is shared by 4 processses;
- The OS creates the illusion that each process executes in its own CPU, i.e. that each process executes in its virtual CPU.
  - When the OS removes the CPU from a process, i.e. the OS preempts the CPU from a process, it must save the CPU state so that it can resume the process later, as if the CPU had not been taken away.
    - This is akin, but not exactly like, a function call, in which the CPU state of the caller is saved in the stack.

States of a Process

- In its lifetime, a process can be in 1 of 3 states:



1. The OS allocates a CPU to the process;
2. The OS allocates the CPU to another process;
3. The process blocks waiting for some event (usually I/O)
4. An event the process was waiting for occurs

Running the CPU executes the process's instructions a executar as instruções do processo;

Waiting the process is waiting for an event (usually the end of an I/O operation)

Ready the process is waiting for the OS to allocate it a CPU, which is executing instructions of another process

Minix 3 Notes: `driver_receive()` is not Polling

`driver_receive()` is a blocking call.

If the process's "IPC queue" is empty:

- The OS will move the process to the WAIT state
- The process' state will be changed to READY, only when a message (or notification) is sent to the process

```
while( 1 ) { /* You may want to use a different condition
    /* Get a request message. */
    if ( driver_receive(ANY, &msg, &ipc_status) != 0 ) {
        printf("driver_receive failed with: %d", r);
        continue;
    }
    if (is_ipc_notify(ipc_status)) { /* received notificat
        switch (_ENDPOINT_P(msg.m_source)) {
        case HARDWARE: /* hardware interrupt notification
            if (msg.NOTIFY_ARG & irq_set) { /* subscribed
                ... /* process it */
            }
            break;
        default:
            break; /* no other notifications expected: do
        }
```

Further Reading

- Sections 2, 2.1
  Andrew Tanenbaum, *Modern Operating Systems*, 2nd Ed.