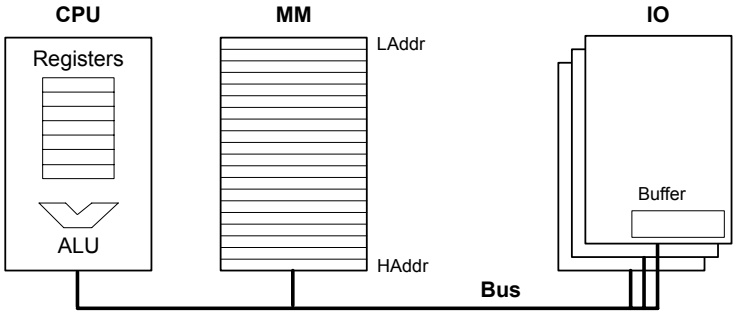
Memory Model Tutorial II : processes and threads

Prologue. The current explanations are very rudimentary and will be completed in a nearby future. An exhaustive oral explanation will be given in class. The figures however are exhaustive.

3.a Main Components of a Computer

Figure 3.1 illustrates the four main components of a classical computer : CPU, Main Memory, IO devices, connected by a Bus.



- CPU: Central Processing Unit
  - ALU: Arithmetic Logic Unit
- MM: Main Memory
  (volatile, i.e. the bits are lost if power is off)
  - Laddr: lowest address
  - Haddr: highest address
- IO: Input / Output
  - display, keyboard, disc, …

Fig. 3.1 Main Components of a Computer

3.c Multiprogram Memory Model

Figure 3.2 illustrates on the left side a classical multiprogram memory model with three loaded prgrams: an Operating System (OS), two programs P1 and P2 and an unused memory slot. The right part of this figure illustrates some important memory details of a program: data, stack and text segments.

An important feature is that the address space of a program is protected, i.e. no other program (except the operating system program) can interfer with it.

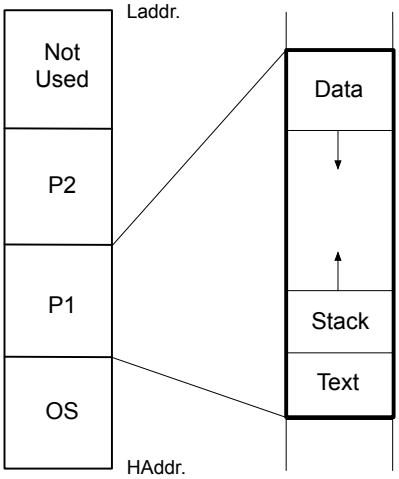


Fig. 3.2 Main Memory: Multiprogram allocation and some details of program P1

3.c Memory Model of a Processs

Program of Fig. 3.3 is illustrated in Fig. 3.4 as a process. All memory parts in the latter figure are located at the main memory module, except for the CPU registers that are located at the CPU chip.

```
char c1, c2;
int fct1() {int i, j; …}
int fct2() {int k; …}

main() {fct1(); fct2();}
```

Fig. 3.3 A simple C program

The **CPU registers** contain two important pointers: a) the stack pointer **SP** which points to the top of the execution stack, and b) the program counter **PC** which points to the next instruction to be executed.

The **Process Control Block** contains all the information about a process that is needed by the operating system : scheduling, resource allocation, interrupt processing, performance monitoring, …

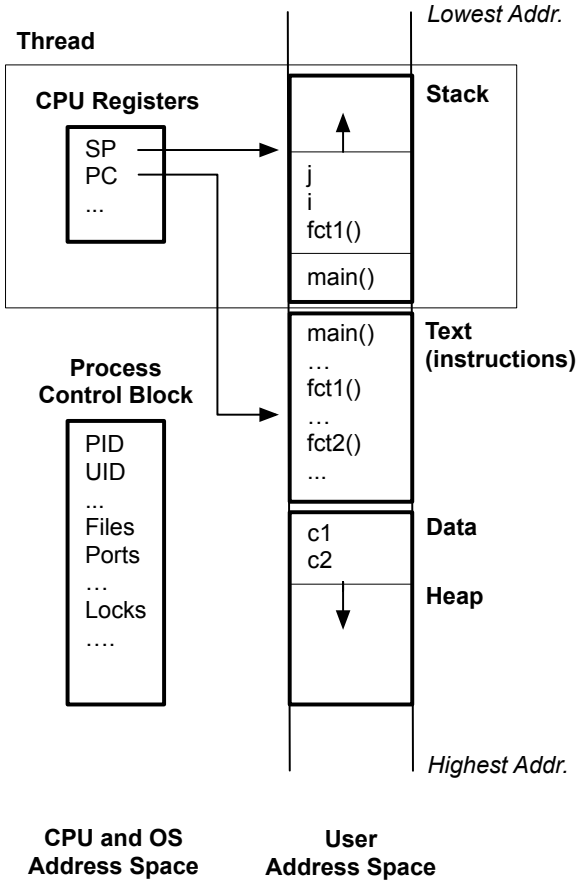


Fig. 3.4 Program of Fig. 3.3 as a process

Remark about multithreading

A generalization of Fig. 3.4 to multithreading is straightforward: just add more threads (and add a Thread Control Bloc per thread, by letting as much stuff as possible in the Process Control Block). Unfortunately neither the programming language constructs are trivial, nor the details of the multithreaded model. Nowadays the most popular ones are Pthreads, a POSIX standard, and SunOS threads. Other models as Amoeba, Chorus, MACH are also of great interest.

Version 0.8, 5 October 2012 (BH)