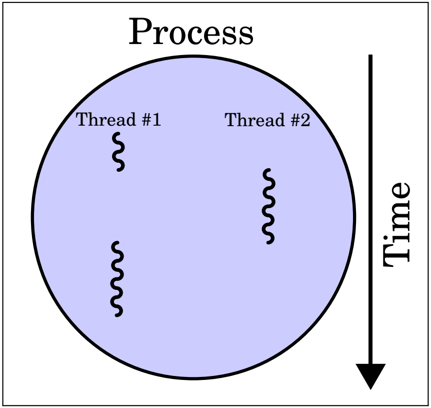
CI583: Data Structures and Operating Systems
OS Design Principles

Outline

Processes and threads

Conceptually, the thread is a unit of computation, to be stopped and started by the OS.



Processes and threads

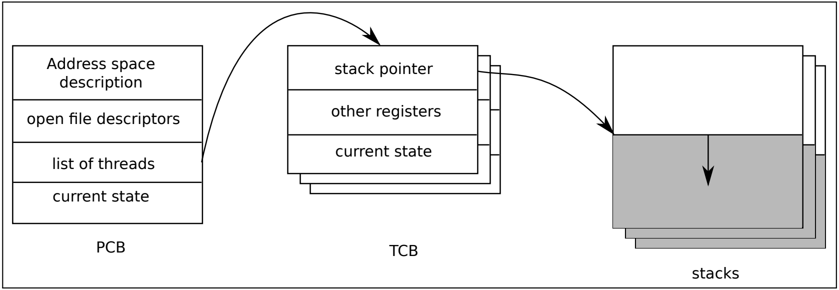
Each thread belongs to an enclosing process, which can also be stopped and started by the OS and which is used to store context which includes:

- an address space (the set of memory locations that contains the code and data for the program),
- a list of references to open files, and
- other information might be common to several threads.

Processes and threads

To represent a process we use a data structure called a process control block (PCB), containing references to all the info given above and to a list of threads.

Each thread is represented by a thread control block (TCB). The TCB contains references to thread-specific context, including the thread's stack and contents of registers.



Processes and threads

The OS needs to be able to create, delete and synchronise processes.

In Windows and *NIX a process is either active or terminated.

Terminated processes are deleted (by a process dedicated to this purpose) when all of its threads are terminated and there are no more references to the process from elsewhere.

Processes and threads

When a process is created, the address space is loaded into memory. On Linux, this is accomplished by the `exec` program.

How does the OS initialise the address space?

One approach would be to copy the code and data of the program into the address space, but this would be inefficient.

Processes and threads

The code section of the program is read-only and so can be shared by any processes executing the same program.

The parts of the data section that are never modified can also be shared.

Processes and threads

A better approach is to map the executable file into the address space.

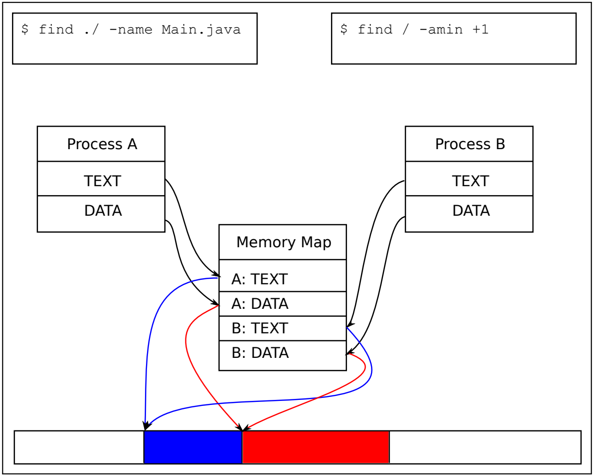
Both the address space and the executable file are divided into blocks of equal size called pages.

Processes and threads

The text regions of all processes running this executable are set up using hardware-address translation facilities, with each process mapping to the same location.

The data regions of each process initially refers to a single copy of the data portion of the executable, which has been copied into memory.

Processes and Threads



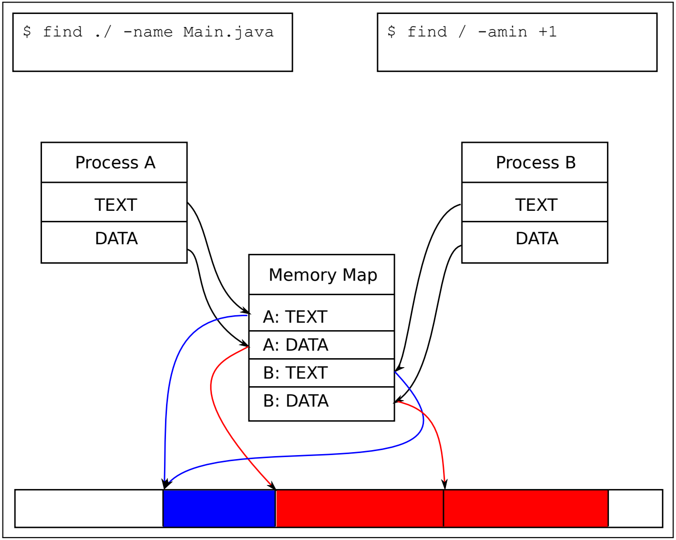
Processes and threads

When a process modifies data for the first time, it is given a new, private page containing a copy of the pristine page.

This is called a private mapping.

Modern systems also use the notion of a shared mapping: when data is modified the original page is altered and all processes see the change.

Processes and threads



Processes and threads

When a thread is created it is in the runnable state.

At some point it is switched into the running state by the scheduler (which we will come back to in more detail).

A thread that is running may then be put into the waiting state, for instance because it has initiated an I/O action and needs to wait for the result before doing anything else.

Processes and threads

The thread can put itself into the waiting state, or can be moved into the state by the OS.

If the OS uses time-slicing, whereby threads are run for a maximum period of time before relinquishing the processor, the scheduler can move the thread into the waiting state.

Processes and threads

A thread can move itself into the terminated state, or it can be moved there by the OS.

Note that a thread must be in the running state at least once before terminating.

Terminated threads are removed in various ways, such as by a dedicated “reaper” process.

Processes and threads