Concurrent Computing: Operating Systems Lecture 5

Multi-Processing - execution occurs in parallel
- eg. multiple cores.

Multi-Tasking - execution occurs concurrently
- eg. one CPU, virtualised to be multiple CPUs

A process is an active instance of a given, passive program image. Each process is made up of:

1. $n \geq 1$ execution contexts each for an independent instruction stream.
2. An associated state, ie. an address space, and a set of resources shared between the execution contexts.

A context switch is the act of changing the active execution context: performing a context switch will typically involve

- Suspending execution of one process, and preserving the execution context in memory.
- loading a new execution context from memory, and resuming execution of that process.

Each process is represented by the kernel, in a process table. Each entry into the table is called a PCB (process control block).

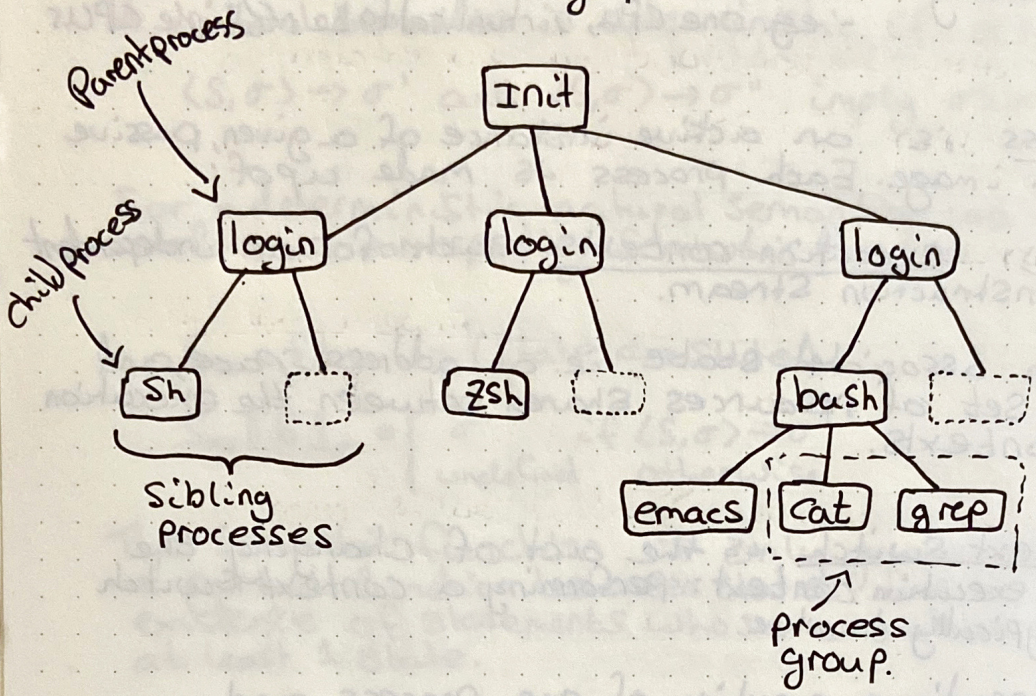
- Entries into the table are very kernel and hardware specific.
- They are divided into per-process and per-thread.

POSIX - representation

POSIX says, processes are organised into a process heirachy, namely a tree, and process groups can be formed to support collective communication.

Here is an example:

- We have 3 logged-in users
- One is executing:
  1. An instance of emacs,
  2. 'cat foo.txt | grep bar'.



As execution progresses, the process ~~state~~ status changes, under control of a scheduler, which tracks processes via scheduling queues eg.:

- 1x ready queue - processes that can be executed.
- nx waiting queues - Processes whose execution is blocked.

POSIX - Creation

A process may be created

- implicitly at boot-time (eg. init)
- explicitly at run-time.

POSIX says we need:

- fork:
  + Create new child process with unique PID.
  + Replicate state, including execution context, address space etc...
  + Return from fork in parent and child processes, st their return values are
    Parent → PID of child.
    Child → 0.
- exec:
  + Replace current process image with new process image.
  + Reset state.
  + No return since call point no longer exists.

A loader performs various tasks related to the latter.

POSIX - Control

POSIX says we need:

- wait:
  + Suspend execution until process (or group) terminates.
  + Recieve error status of said process.
- Sleep:
  -
  + Suspend execution for specified time period.
  + Close to, but not quite yield.

POSIX - termination

A process may terminate due to; exit; controlled error; uncontrolled error or signal.

They can be classified as normal, abnormal and external events.

POSIX says we need:

- exit:
  + Perform normal termination
  + Invoke call backs, flush then close open files
  + Pass exit status to parent process (via a wait).
- abort:
  + Perform abnormal termination.

In both cases the PCB is eventually removed.

Concurrent Computing: Operating Systems - Lecture 6

Scheduler Invocation.

A scheduler is typically classified as:

- A short-term scheduler, is invoked frequently and tasked with selecting a process to execute from the ready queue.
- A long-term scheduler, is invoked infrequently
  - Controlling the degree of multi-processing
  - Ensuring an effective mix of processes.

A multi-tasking kernel may be:

- Pre-emptive, scheduler is forcibly invoked.
- Co-operative, scheduler invoked by currently executing process.

We can invoke the scheduler when:

- Process terminates
- Process intentionally co-operates eg. via yield.
- Process unintentionally co-operates eg. via read.
- A pre-organised interrupt is requested.
  - Fix a time quantum τ.
  - Hardware timer, s.t. an interrupt is requested every τ time units.

continues on page 17.

← Back to page 8.

Scheduler Algorithm.

A Batch System is s.t. all process are specified before execution and complete without interaction from a user. Often CPU-Bound.

An Interactive System is s.t. processes maybe specified before execution, and complete with interaction with a user. Often I/O-Bound.

A real-time system is s.t. all ~~eat~~ processes have a deadline to complete:

- Soft real-time systems are less strict.
- Hard real-time systems are very strict.

A deadline usually occurs due to the need to respond to an event.

The arrival time of a process is deemed to be the point when it enters the ready queue.

Here are some simple algorithms for schedulers to use:

- Random
- Round Robin
- First come first served.
- Shortest job first.
- Priority-based.

Multilevel Queue Scheduling:

- L separate queues, potentially managed using different algorithms.
- Use a scheduling class to assign each process to a level upon arrival.
- Select a Pi from the highest non-empty level
- Allow processes to migrate between levels.