Managing Processes

Dr Tran Duc Minh and Dr Hung Tran

DATCOM Lab
Faculty of Data Science and Artificial Intelligence
College of Technology, National Economics University
Email: minhdt@neu.edu.vn, hung.tran@neu.edu.vn

March 24, 2025

Outline

Introduction

- **A process** is a running program.
  - We use the term process rather than program because a **program is a static entity** while a process has a **state** which changes over time.
- In Linux, processes can be started from the GUI or the command line or by other running processes.
  - Processes that run with user interaction are known as **foreground processes** while those that run with no interaction are **background processes**.
- Whenever a process runs, the Linux kernel keeps track of it through a **process ID** (PID).
  - After the Linux kernel is loaded and running, the first process it launches is called **systemd**. **systemd** is responsible for starting the run-time environment and then monitoring the environment while in use. systemd is given a PID of 1.

Introduction

- In Linux, a process can only be created by another process (with the exception of systemd).
  - We refer to the creating process as the parent and the created process as the child whereby the parent process spawns the child process.

```
systemd
      `-- gnome
            `-- gnome_terminal
                  `--bash
                        `--./somescript
                        `--find
```

Example of parent/child relationships in Linux.

Forms of process management

- **a program** is a piece of software.
  - The program exists in one of two states: the original source code and the executable code. In either case, the program is a **static entity**.
- The process, on the other hand, is an **active entity**.
  - As the process executes, the values it stores in memory (variables) change. As the process executes, it may request resources from the operating system which it then holds, uses and returns to the operating system.
- We refer to **the running program as a process** to differentiate it from the static program.
  - We can have multiple copies of the same program running at a time. The programs are identical while the processes all differ because each will have its own unique state.

Forms of process management

- The CPU stores information about the running process via its registers.
  - **The program counter (PC)** stores the address of the next instruction to fetch from memory.
  - **Instruction register (IR)** stores the current instruction.
  - **Data registers** store data currently in use.
  - The results of the most recent operation are indicated using status flags in the status **flag (SF) register**.
  - A run-time stack is maintained for this process, indicating subroutine invocation and parameter passing. The top of the stack is pointed to by **the stack pointer (SP)**.

Forms of process management

- The operating system must keep track of the process's state and does so through a data structure often called the **process control block** (PCB).
  - The PCB is a collection of the most important pieces of information about the running process.

Process Control Block Information

Item	Use
Process state	The run-time state of the process; usually one of `new`, `ready`, `running`, `suspended`, `terminating`, `waiting`.
Process ID	Usually a numeric designator to differentiate the process from others.
Other process data	Parent process (if there is one), process owner, priority and/or scheduling information, accounting data such as amount of CPU time elapsed.
Process location	Which queue the process currently resides in (ready queue, wait queue, job queue).
Process privilege/state	What mode the process runs in (user, privileged, some other).
Hardware-stored values	PC, IR, SF, SP (and possibly data register values) and interrupt masks.
Resource allocation	I/O devices currently allocated to the process; memory in use; page table.

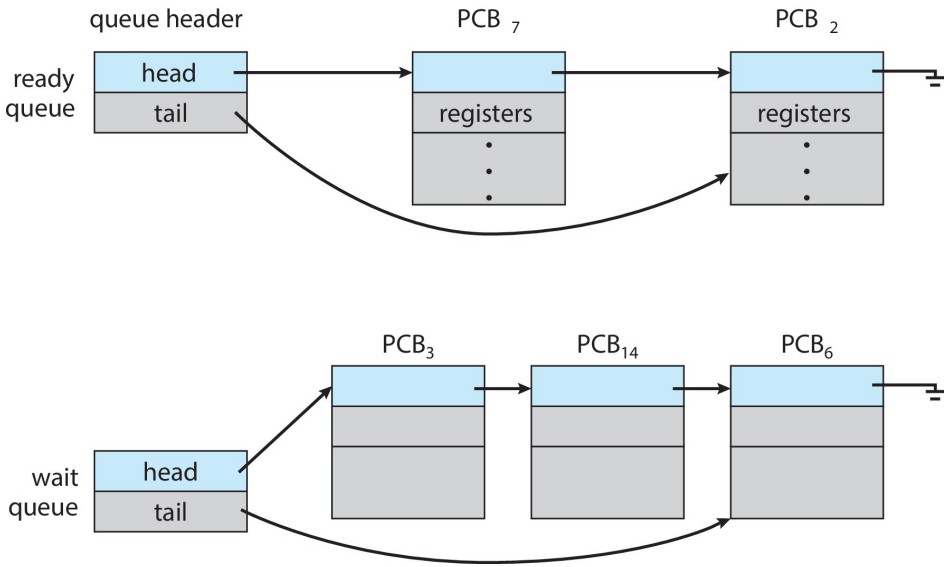
- Aside from keeping track of a process' status, the operating system is in charge of scheduling when processes run, and of changing the process' status as needed.

Single-Process Execution

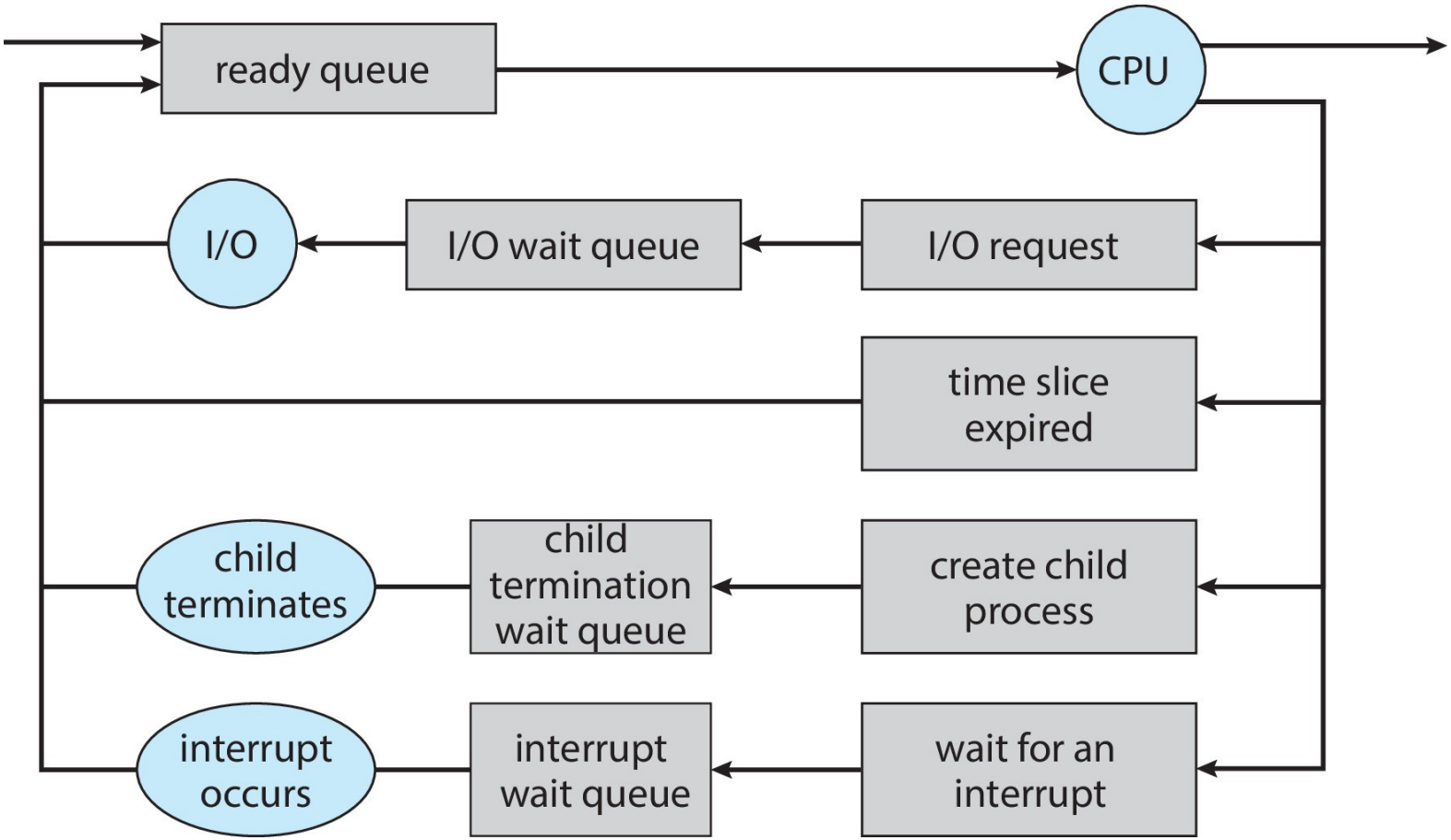
- In **single tasking**, the operating system starts a process. The CPU executes this one process until either the process terminates or requests some operation by the operating system.
  - In the latter case, the CPU switches context from the running process to the operating system. Upon completion of the request, the context switches back to the process. This means that **the computer is limited to running only one process at a time**.
- Single tasking is the easiest form of process management to implement but has many drawbacks.
  - The user is limited to running one task at a time.
  - If the process has to perform some time-consuming input or output, the CPU remains **idle** while the input/output (I/O) takes place.

Concurrent Processing

- One of the great inefficiencies of both single tasking is that of I/O. It would be nice to give the CPU something to do while waiting.
- We have at least two queues. One is the ready queue which contains those processes that are ready for execution; the other is a wait queue where processes are moved when they are waiting for I/O.

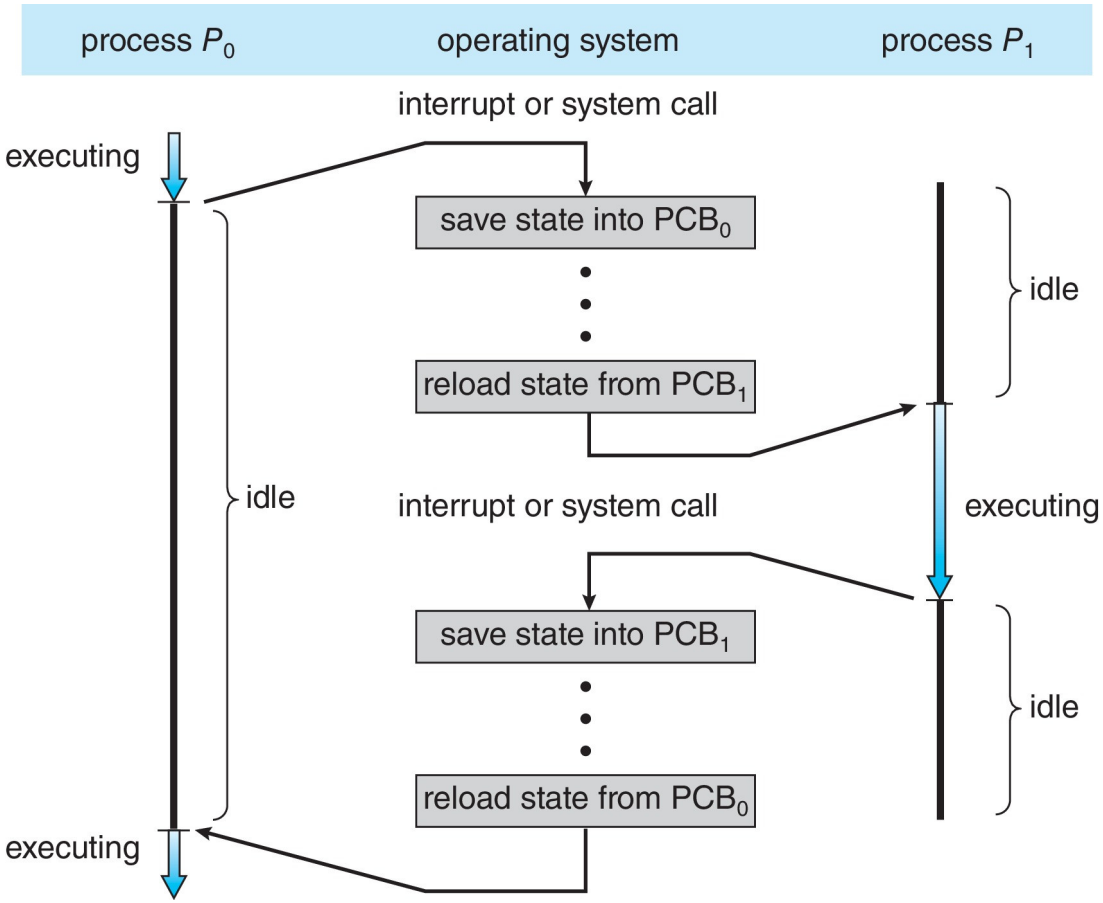


Concurrent Processing



CPU Switch From Process to Process

- A **context switch** occurs when the CPU switches from one process to another.



Linux System Calls to Create Processes

Linux System Calls to Create Processes

System Call	Result of the Call
`clone()`	Like `fork` (see below) except there is more control over the child process produced with respect to what is duplicated and what is shared between parent and child.
`exec()`	Take an existing process and replace its image (executable code) with a new image.
`fork()`	Create a duplicate process of the parent but with its own PID, its own memory and its own resources; parent and child can run concurrently.
`vfork()`	Same as `fork` except parent is temporarily suspended and child might be permitted to use the parent's memory space.
`wait()`	Suspend parent process to wait for an event of a child process.

Questions