University of Glasgow | School of Computing Science

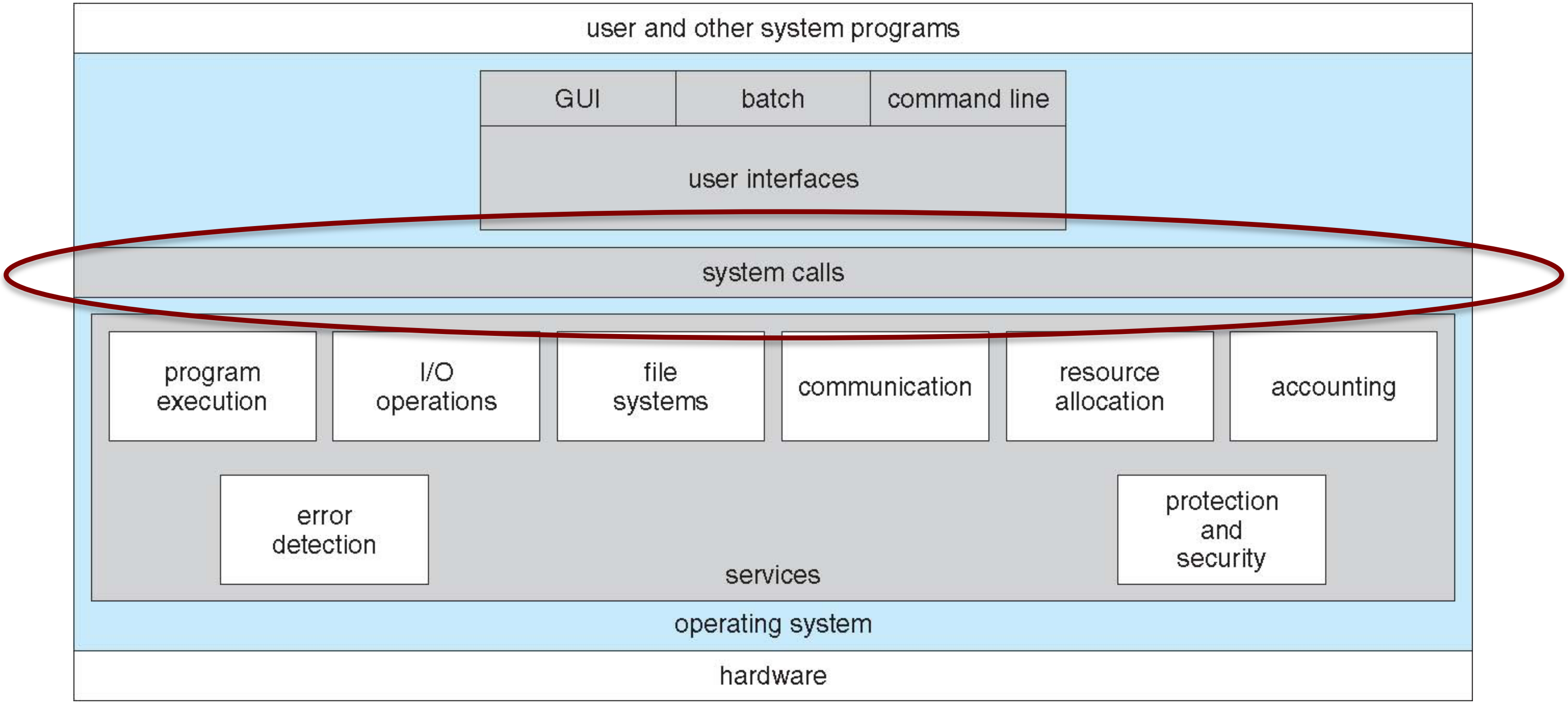
Networks & Operating Systems Essentials

Dr Angelos Marnerides

<angelos.marnerides@glasgow.ac.uk>

School of Computing Science

Operating System Services



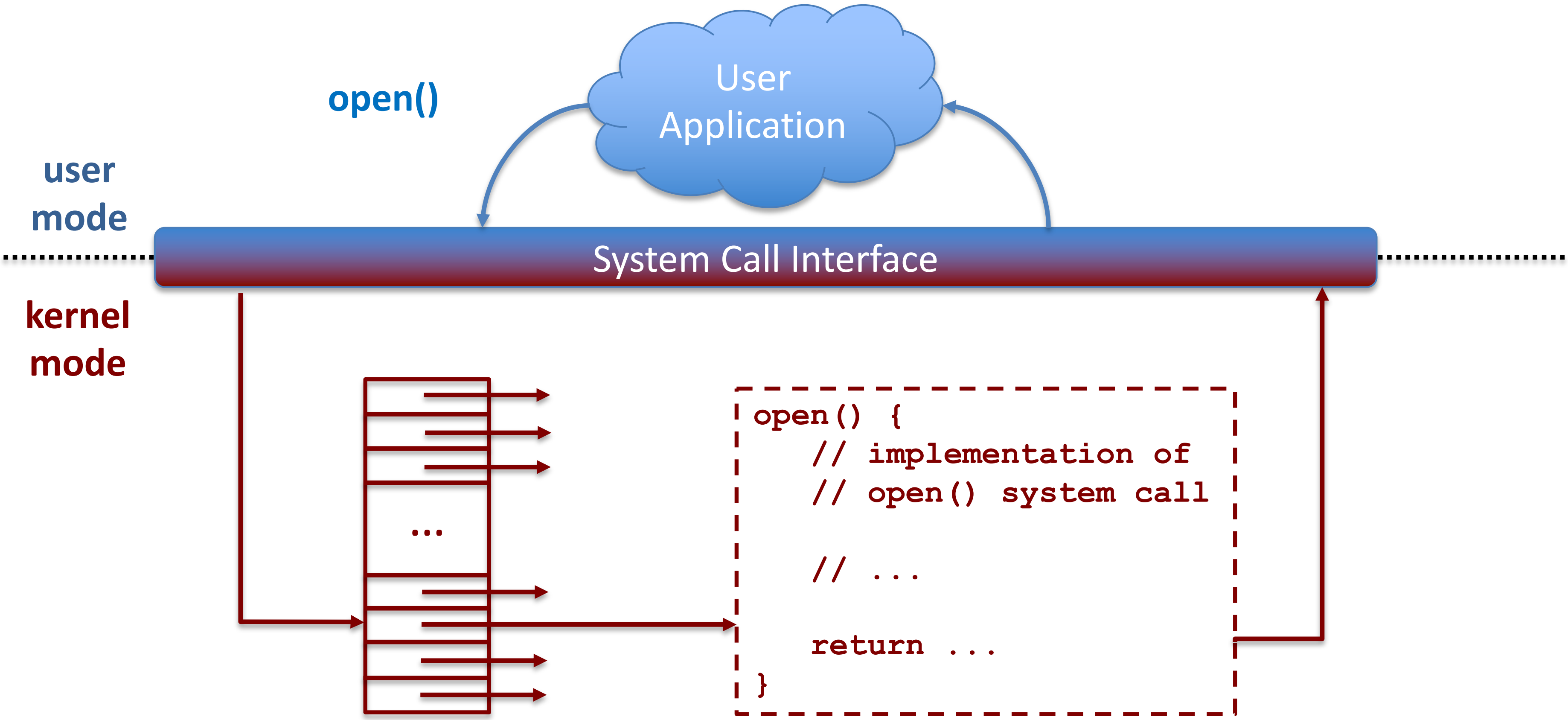
System Calls

- Programming interface to the services provided by the OS
  - Typically written in a high-level language (C or C++)

- Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use

- Use APIs or system calls?

System calls



open()

User Application

user mode

System Call Interface

kernel mode

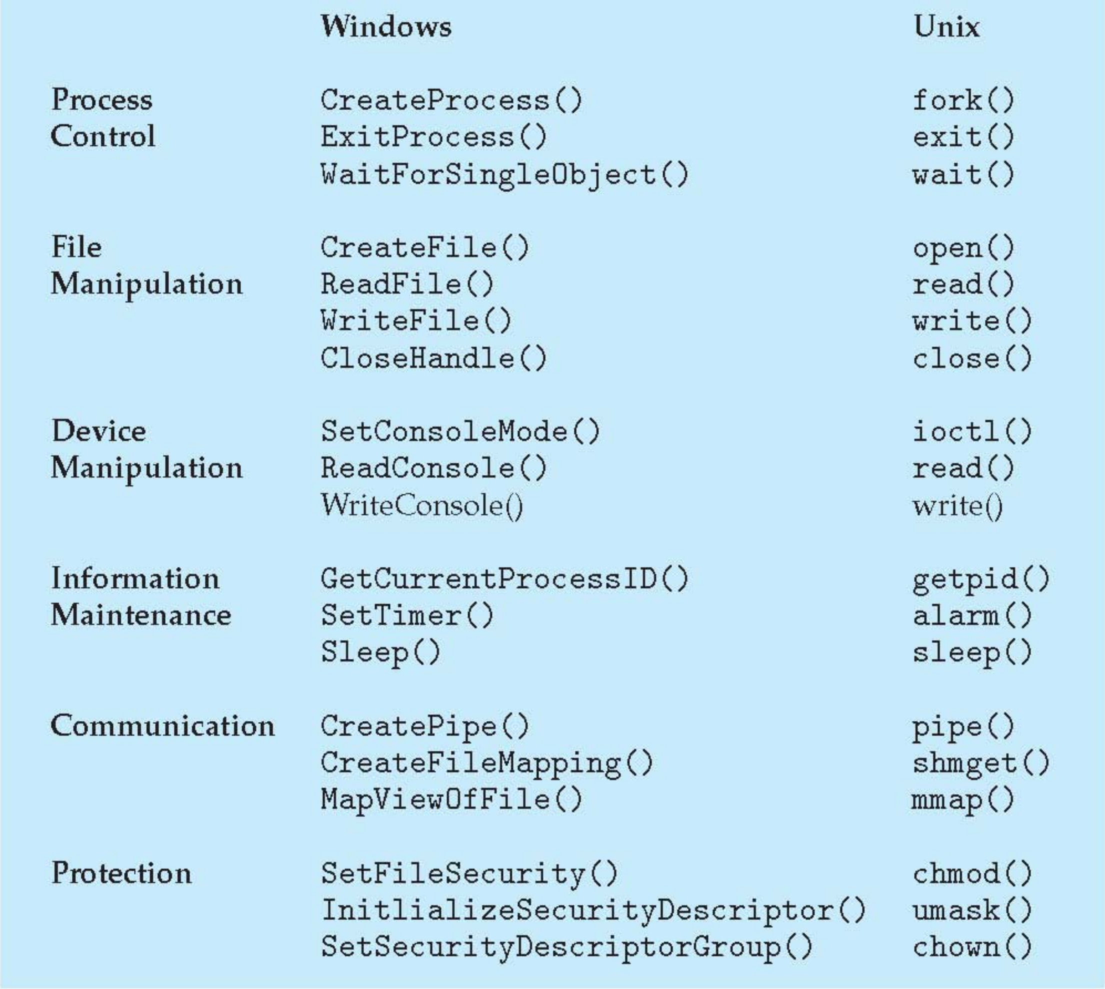
...

```
open() {
    // implementation of
    // open() system call

    // ...

    return ...
}
```

System Call Parameter Passing

- Methods for passing parameters (to the OS)?
  - Pass the parameters in registers
  - Parameters stored in memory (block or table), and address of block passed as a parameter in a register
  - Placed/pushed, onto the stack by the program and popped off the stack by the operating system

Examples of Windows and Unix System Calls



	Windows	Unix

Process Control	CreateProcess()
	ExitProcess()
	WaitForSingleObject()
		fork()
		exit()
		wait()

File Manipulation	CreateFile()
	ReadFile()
	WriteFile()
	CloseHandle()
		open()
		read()
		write()
		close()

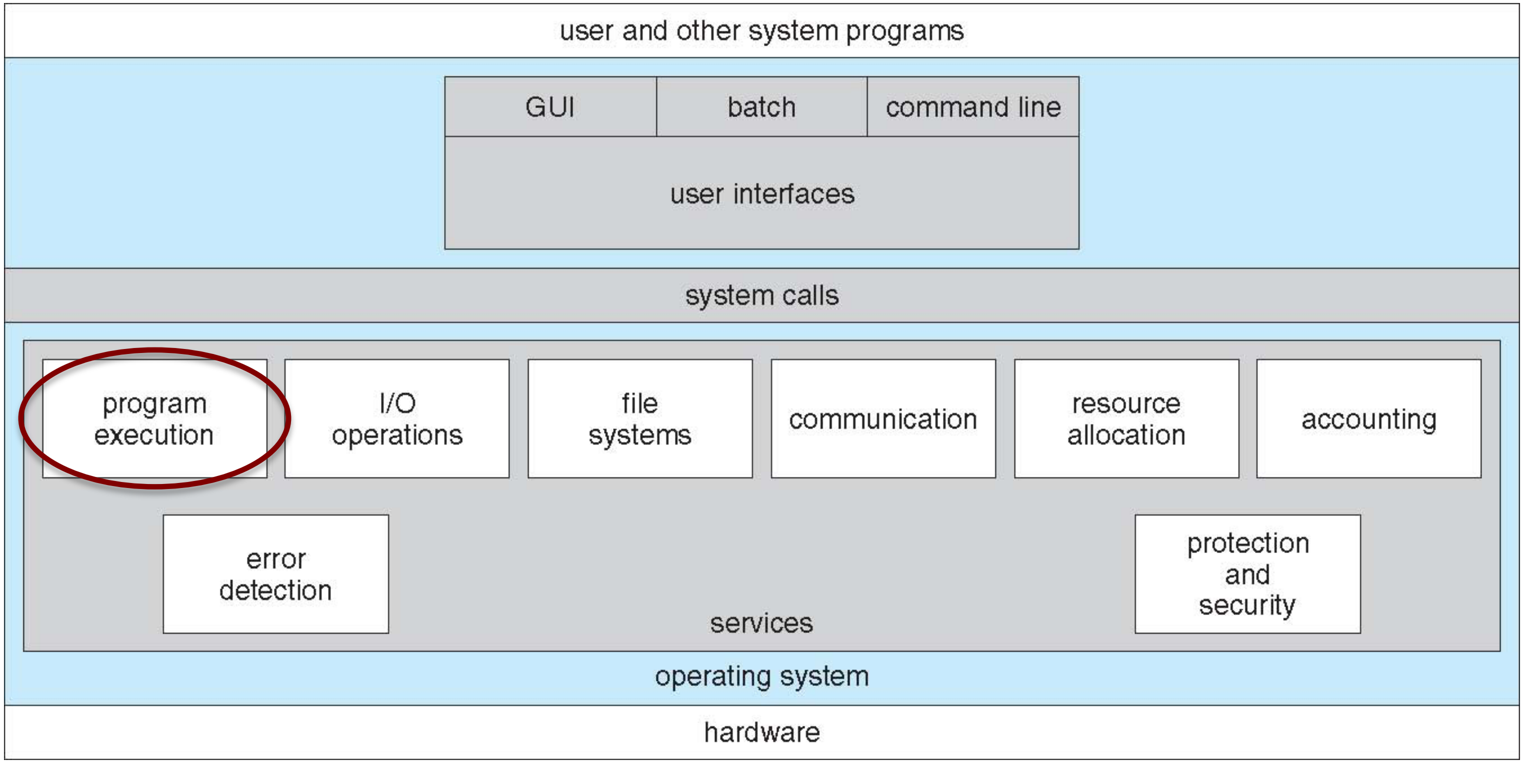
Device Manipulation	SetConsoleMode()
	ReadConsole()
	WriteConsole()
		ioctl()
		read()
		write()

Information Maintenance	GetCurrentProcessID()
	SetTimer()
	Sleep()
		getpid()
		alarm()
		sleep()

Communication	CreatePipe()
	CreateFileMapping()
	MapViewOfFile()
		pipe()
		shmget()
		mmap()

Protection	SetFileSecurity()
	InitlializeSecurityDescriptor()
	SetSecurityDescriptorGroup()
		chmod()
		umask()
		chown()

Operating System Services



Process In general

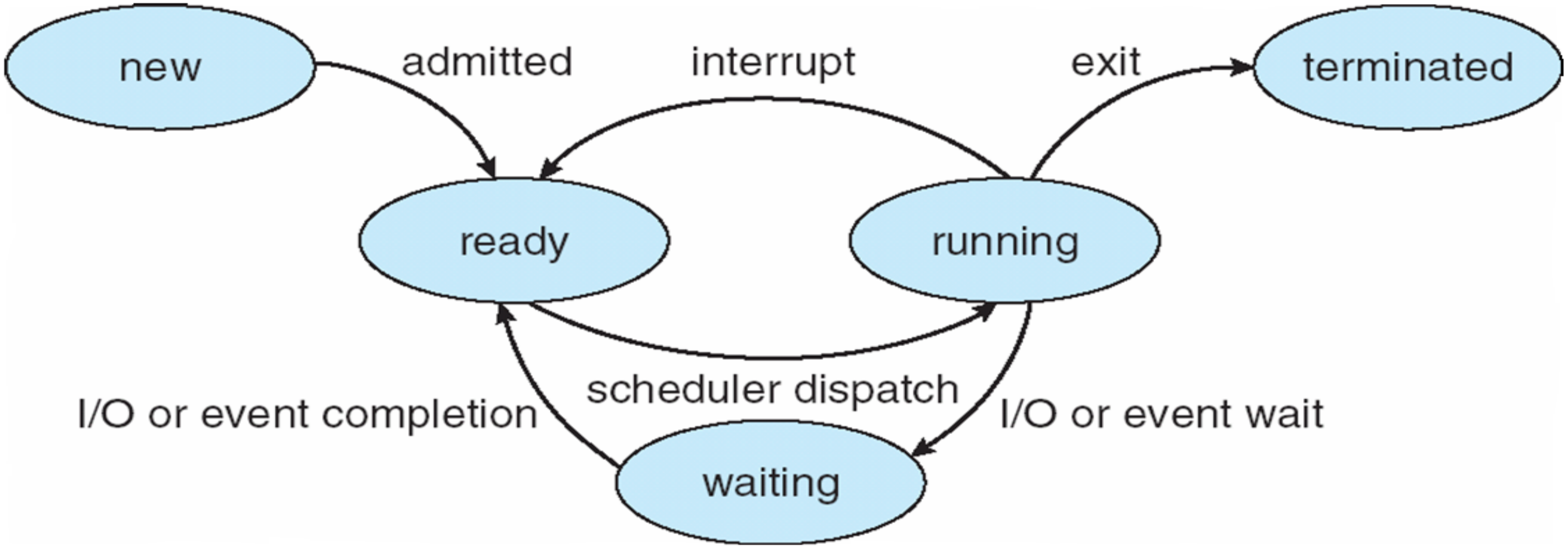
- Process – a program in execution
  - Program is passive entity stored on disk (executable file)
  - A program becomes a process when executable file loaded into memory
  - One program can be several processes

Multi-process example

- Many web browsers ran as single process (some still do)

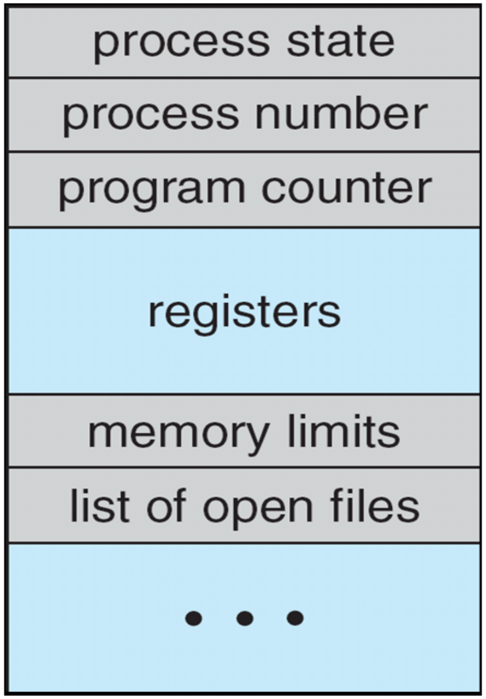
- Google Chrome Browser is multi-process:
  - Browser process manages user interface, disk and network I/O
  - Renderer process
  - Plug-in process for each type of plug-in

Diagram of Process States



Process Control Block (PCB)

- Information associated with each process
  - Also known as Process Table Entry



process state

process number

program counter

registers

memory limits

list of open files

• • •

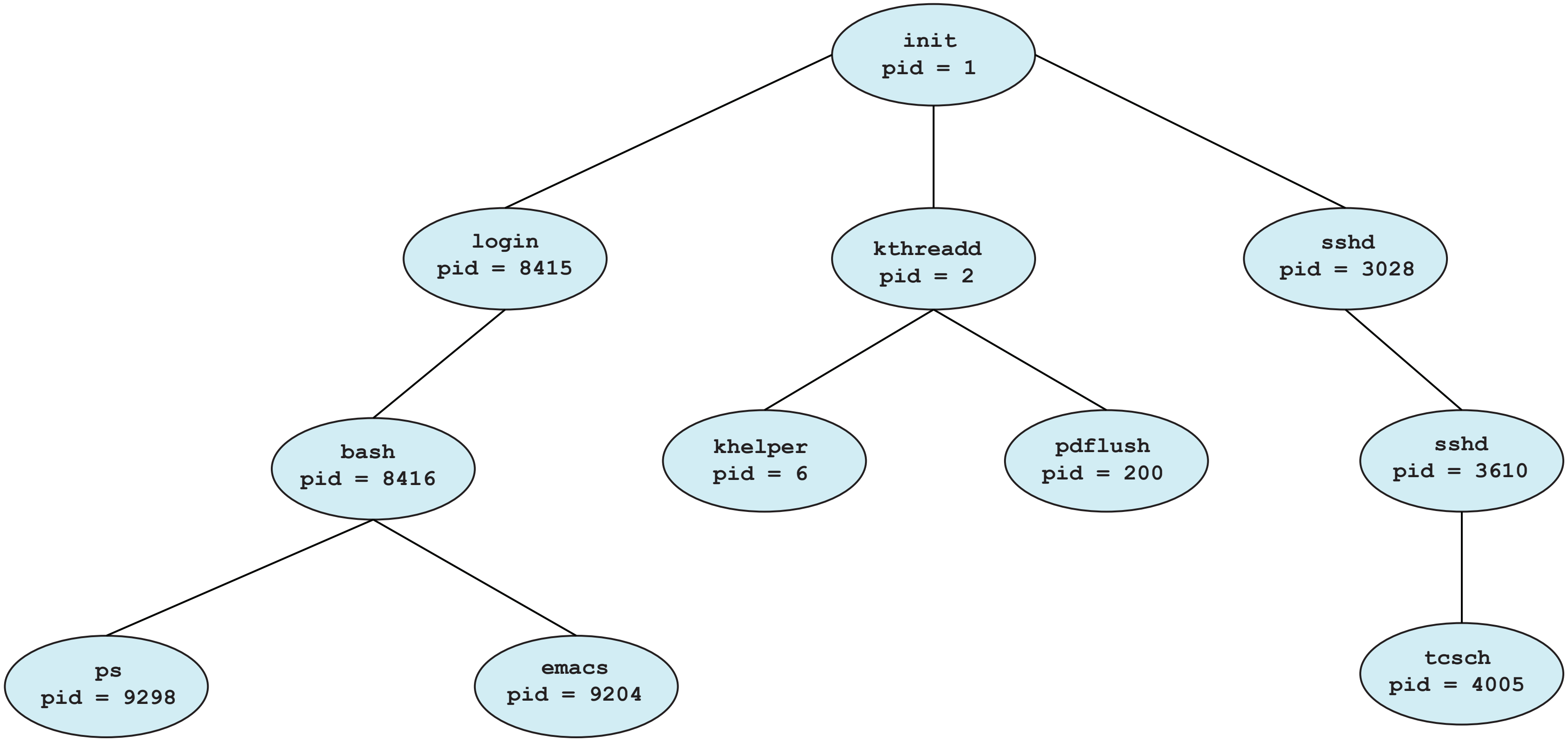
Process Creation

- Parent process → child processes
  - A “process tree”
  - *`fork(2)`*, *`exec(3)`*, *`wait(2)`*

- Execution options
  - Parent and children execute concurrently
  - Parent waits for children to terminate

- Resource sharing options
  - Parent and children share all resources
  - Children share subset of parent’s resources
  - Parent and child share no resources

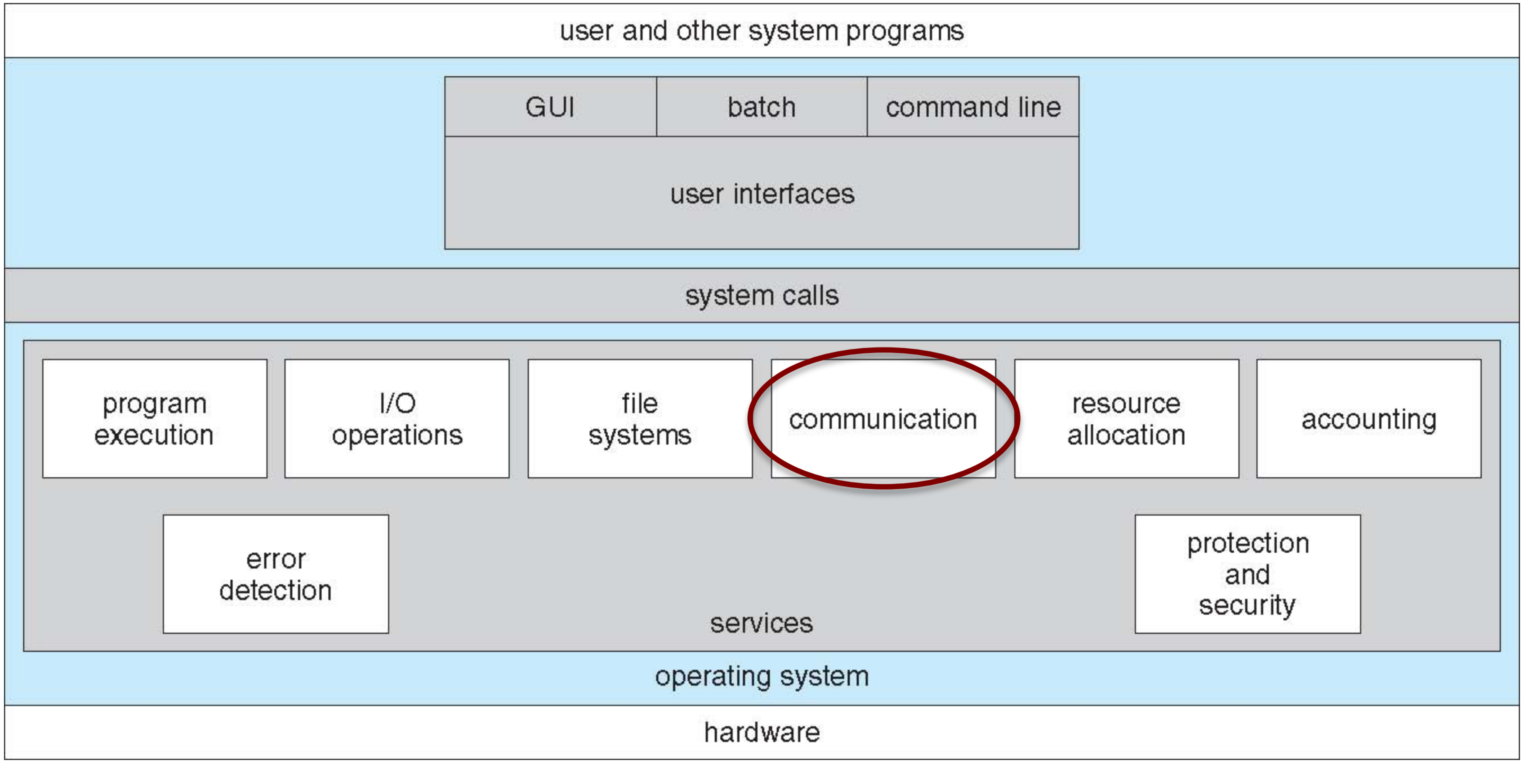
Linux process tree



Aside: Is there life after death…

- A parent process must wait for child processes to terminate, then “reap” (collect and free) their state
  - Failure to do so creates zombie processes
- If a parent process terminates before one of its child processes?
  - The child process becomes an orphan processes

Operating System Services



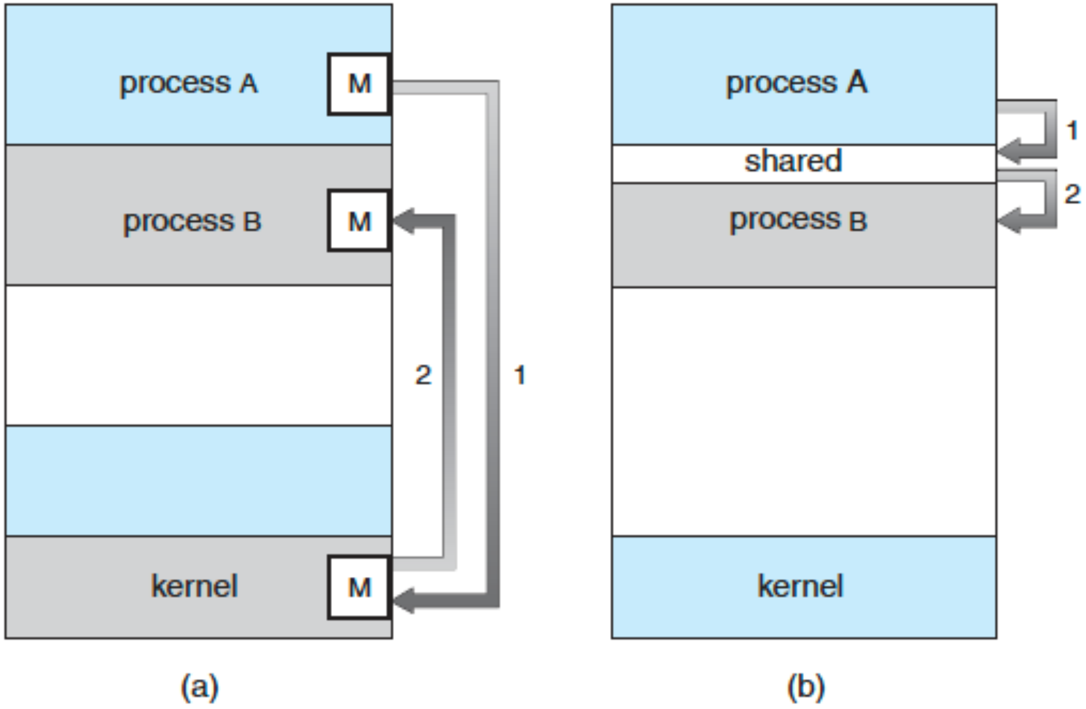
Inter-process Communication (IPC)

- Processes may be independent or cooperating
- Cooperating process can affect or be affected by other processes, including sharing data

- Advantages of cooperating processes?
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience

Communications Models

- Two models of IPC
  - (a) Message passing
  - (b) Shared memory



Message Passing

- Requires a “communication link” between communication processes
- Provides at least two operations: `send(message)` and `receive(message)`
- Considerations:
  - Direct vs indirect communication
  - Synchronous vs asynchronous communication
  - Automatic vs explicit buffering

Message Passing: Direct Communication

- Processes must name each other explicitly (symmetric):
  - send (P, message) – send a message to process P
  - receive(Q, message) – receive a message from process Q
- Alternatively, only the recipient needs to be explicitly named (asymmetric)
  - send (P, message) – send a message to process P
  - receive(id, message) – receive a message from any process (id set to the name of the sender on reception of a message)

- Communication links are established automatically
- Exactly one link between a pair of communicating processes
- Links may be unidirectional, but usually are bi-directional

Message Passing: Indirect Communication

- Messages are directed and received from mailboxes
  - Sometimes mentioned as ports
  - Not to be confused with Transport Layer ports
- Processes can communicate only if they share a mailbox
- Links established only if processes share a common mailbox
- A link may be associated with many processes
- Each pair of processes may share several communication links that may be unidirectional or bi-directional

Message Passing: Indirect Communication

- OS must provide mechanisms to:
  - Create a new mailbox
  - Send and receive messages through mailbox
  - Destroy the mailbox
- At first, the creator/owner of a mailbox is the only recipient
  - Ownership/receiving privilege can be passed to/shared with other processes through system calls
- Interface:
  - send(A, message) – send a message to mailbox A
  - receive(A, message) – receive a message from mailbox A

Message Passing: Synchronization

- Blocking (synchronous)
  - Blocking send → sender blocked until message received
  - Blocking receive → receiver blocked until message available

- Non-blocking (asynchronous)
  - Non-blocking send → sender enqueues the message and continues operation
  - Non-blocking receive → receiver receives either a valid message or a null

Message Passing: Buffering

- Exchanged messages go through a temporary queue (buffer)
- How large is this buffer?
  - Zero capacity – 0 messages
    - Sender must wait for receiver
  - Bounded capacity – finite length of n messages
    - Sender must wait if link full
  - Unbounded capacity – infinite length
    - Sender never waits

Communication Models (revisited)

- Two models of IPC
  - Message passing
  - Shared memory
- Will the above work for client-server applications?
- Will they work for processes on different machines?
- Enter alternatives:
  - Sockets
  - Remote Procedure Call (RPC)
  - Pipes

Recommended Reading

- Gagne, Silberschatz and Galvin, Operating Systems Essentials (2011), Chapter 3, Sections 3.3 and 3.4.