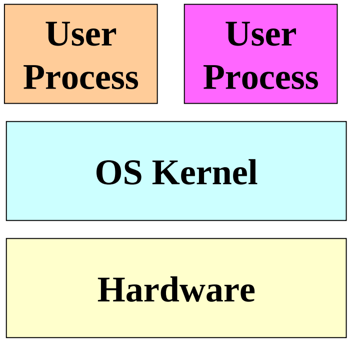
Processes

CS 217

Operating System

- Supports virtual machines
  - Promises each process the illusion of having whole machine to itself
- Provides services:
  - Protection
  - Scheduling
  - Memory management
  - File systems
  - Synchronization
  - etc.



User Process

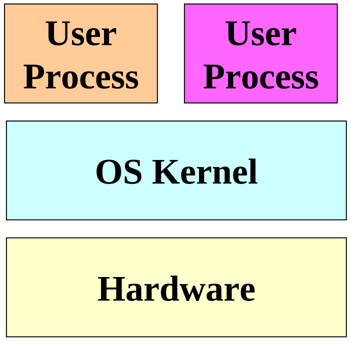
User Process

OS Kernel

Hardware

What is a Process?

- A process is a running program with its own …
  - Processor state
    - EIP, EFLAGS, registers
  - Address space (memory)
    - Text, bss, data, heap, stack



User Process

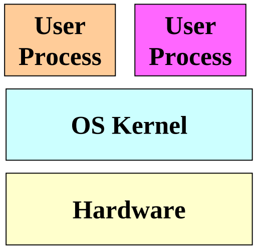
User Process

OS Kernel

Hardware

Operating System

- Resource allocation
  - Sharing
  - Protection
  - Fairness
  - Higher-level abstractions



User Process

User Process

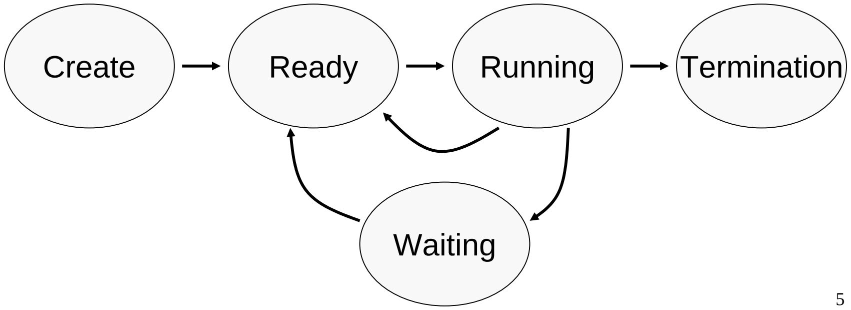
OS Kernel

Hardware

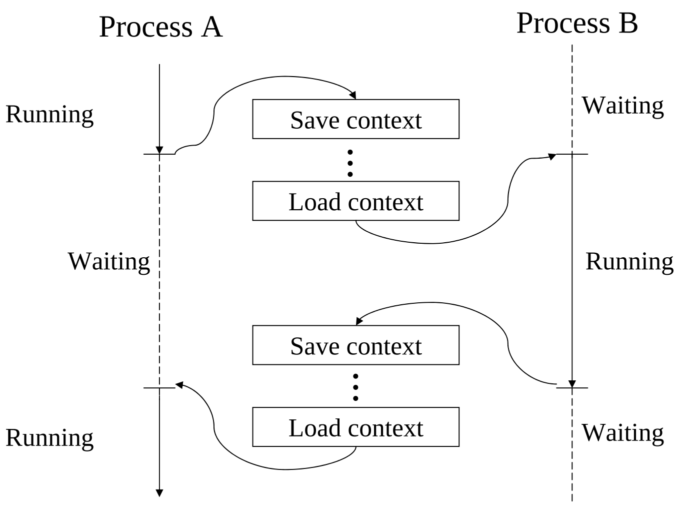
- Common strategies
  - Chop up resources into small pieces and allocate small pieces at fine-grain level
  - Introduce level of indirection and provide mapping from virtual resources to physical ones
  - Use past history to predict future behavior

Life Cycle of a Process

- Running: instructions are being executed
- Waiting: waiting for some event (e.g., i/o finish)
- Ready: ready to be assigned to a processor

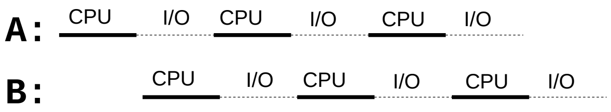


Context Switch



Overlap CPU with I/O operations

- Schedule CPU for process B while process A is waiting for I/O
  - Better utilize CPU



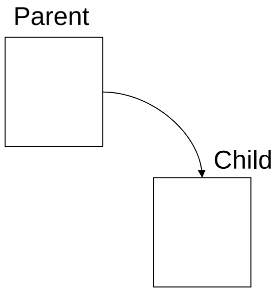
Process Control Block

- For each process, the kernel keeps track of ...
  - Process state (new, ready, waiting, halted)
  - CPU registers (EIP, EFLAGS, EAX, EBX, …)
  - CPU scheduling information (priority, queues, ...)
  - Memory management information (page tables, ...)
  - Accounting information (time limits, group ID, ...)
  - I/O status information (open files, I/O requests, ...)

Fork

- Create a new process (system call)
  - child process inherits state from parent process
  - parent and child have separate copies of that state
  - parent and child share access to any open files

```
pid = fork();
if (pid != 0) {
    /* in parent */
    ...
}
/* in child */
...
```



Wait

- Parent waits for a child (system call)
  - blocks until a child terminates
  - returns pid of the child process
  - returns –1 if no children exist (already exited)
  - `status`

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int *status);
```

- Parent waits for a specific child to terminate

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t waitpid(pid_t pid, int *status, int options);
```

Fork

- Inherited:
  - user and group IDs
  - environment
  - close-on-exec flag
  - signal handling settings
  - supplementary group IDs
  - set-user-ID mode bit
  - set-group-ID mode bit
  - profiling on/off/mode status
  - debugger tracing status
  - nice value
  - stdin
  - scheduler class
  - all shared memory segments
  - all mapped files
  - file pointers
  - non-degrading priority
  - process group ID
  - session ID
  - current working directory
  - root directory
  - file mode creation mask
  - resource limits
  - controlling terminal
  - all machine register states
  - control register(s)
- Separate in child
  - process ID
  - address space (memory)
  - file descriptors
  - active process group ID.
  - parent process ID
  - process locks, file locks, page locks, text locks and data locks
  - pending signals
  - timer signal reset times
  - share mask

Exec

- Overlay current process image with a specified image file (system call)
  - affects process memory and registers
  - has no affect on file table
- Example:

```
execlp(“ls”, “ls”, “-l”, NULL);
fprintf(stderr, “exec failed\n”);
exit(1);
```

Exec (cont)

- Many variations of **exec**

```
int execlp(const char *file,
           const char *arg, ...)
int execl(const char *path,
           const char *arg, ...)
int execv(const char *path,
           char * const argv[])
int execle(const char *path,
           const char *arg, ...,
           char * const envp[])
```

- Also **execve** and **execvp**

Fork/Exec

- Commonly used together by the shell

```
... parse command line ...
pid = fork()
if (pid == -1)
   fprintf(stderr, “fork failed\n”);
else if (pid == 0) {
   /* in child */
   execvp(file, argv);
   fprintf(stderr, “exec failed\n”);
}
else {
   /* in parent */
   pid = wait(&status);
}
... return to top of loop ...
```

System

- Convenient way to invoke fork/exec/wait
  - Forks new process
  - Execs command
  - Waits until it is complete

```
int system(const char *cmd);
```

- Example:

```
int main()
{
    system(“echo Hello world”);
}
```

Summary

- Operating systems manage resources
  - Divide up resources (e.g., quantum time slides)
  - Allocate them (e.g., process scheduling)
- A processes is a running program with its own …
  - Processor state
  - Address space (memory)
- Create and manage processes with ...
  - **fork**
  - **exec**
  - **wait**
  - **system**

} Used in shell