

```

Give an example of a compile time error.  int *n;
Give an example of a link time error.    scan( "%i", n )
Give an example of a runtime error.     scan( "%i", n );
                                           scanf( "%i", n );

```

What's the difference between a program and a process?

Program: executable instructions (and possibly data) in a disk file.
 Process: program loaded into memory.

What are the parts of a program image on disk?

text, initialized data

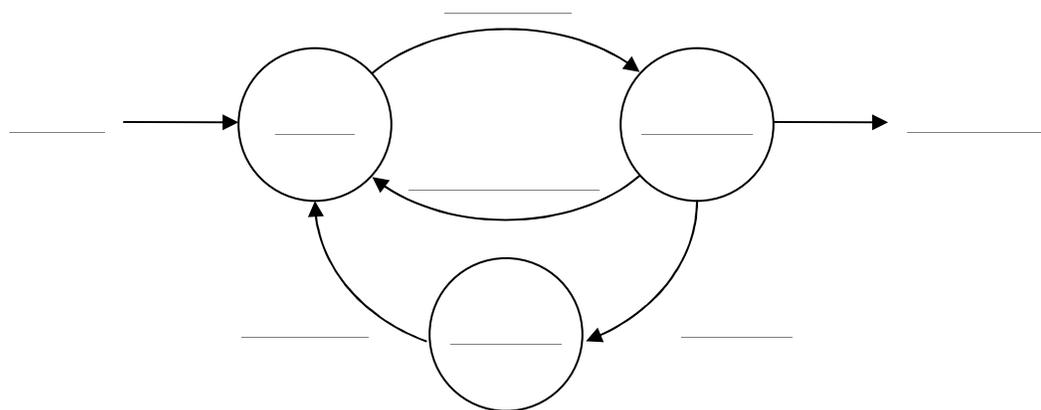
What are the parts of a process image in memory?

text, (initialized) data, BSS (uninitialized data), heap, stack

What do we mean by multiprogrammed ?

A system supporting multiple processes that can share resources.

Label this process state diagram:



Describe these signals:

SIGCHLD	sent to parent process when child terminates
SIGINT	sent by shell to foreground process when ^C entered
SIGUSR1	user-defined
SIGKILL	terminates a process. Cannot be caught or ignored
SIGTERM	terminate a process 'politely': can be caught and ignored.
SIGALRM	sent by OS when timer expires

All can be sent by the shell: \$ kill -INT pid
 \$ kill -SIGINT pid
 \$ kill -2 pid

In what ways can a user process can receive information ...

From the shell via **command-line arguments**.

example: `$ mmame -min -max`

From a **resource** file when the process starts up.

example: `/bin/bash` reads `.bashrc`

From a **file**.

example: read from a regular file or a pipe

From the OS via a **system call return value**.

example: `fork()` returns `-1` (failure), pid of child, or `0`

From the OS via **system call parameters**.

example: `struct stat s;`
`stat("foo", &s);`

From the OS via shell **environment variables**.

example: `getenv("PWD")`

From the OS via a **signal**.

example: SIGFPE received on divide-by-zero

From another process via a signal.

example: `kill(getpid(), SIGUSR1);`

From another process as a result of a **fork()**:

example: child gets parent's open file descriptors

example: child gets a copy of parent's in-scope variables

From another process via **shared variables**.

From another process via a **semaphore**.

example: synchronization information - process can proceed past a
`wait(s)`

In what ways can a user process can pass information to the OS?

Return value from function main(): `return 0;`

By passing values in a system call or library function call:

example: `_exit(value);` // system call

example: `exit(value);` // library function

example: `printf("%i", n);` // library function

Through **environment variables:**

example: `putenv("PATH=thisdir");`

`execlp("foo", "foo", NULL);`

What is a race condition?

In a multi-programmed system, a condition where behavior depends on execution order.

Why are race conditions possible?

The order of execution in a multiprogrammed system is non-deterministic: context switches can occur at any time, due to:

User process gives up the CPU intentionally:
 pause() system call - wait for any signal
 sleep(), usleep() - wait for SIGALRM
User process makes an IO request, causing it to block
OS scheduling algorithm: timer runout
Hardware traps: e.g., illegal instruction, divide-by-zero

What is an atomic operation?

One that can't be interrupted: it will complete before a context switch can occur

What is a synchronization primitive?

A mechanism for ordering execution that relies on an atomic operation.

What do we mean by "cooperating processes"

Multiple processes that must coordinate their execution sequence, or that have access to shared resources.

What is a critical section?

A section of code that accesses resources shared by cooperating processes.

What is mutual exclusion?

A requirement that only one process at a time is allowed access a shared resource.

What is the critical section (CS) problem?

That of ensuring mutual exclusion.

What's the general structure of cooperating processes?

```
ENTRY          <- code indicating a process wants to enter its CS
    critical section
EXIT           <- code indicating a process is has left its CS
remainder
```

What is required for a valid solution to the critical section problem?

Mutual exclusion - only one process at a time can be executing in a critical section
Progress - a process not in a critical section can't keep another out of a critical section
Bounded wait - there is an upper limit on the time a process can be held in its ENTRY section

In general, the critical section problem may be solved by what means?

Algorithm (software) - e.g., Dekker, Peterson
Synchronization primitive
Disabling interrupts

How can disabling interrupts be used to solve the critical section problem?

Mutual exclusion from a critical section requires that only one process at a time can be executing in its critical section. Disabling interrupts prevents context switching. So for the duration that interrupts are disabled, no other process can enter its critical section, since no other process can run.

Why is it a bad idea to let a user-process disable interrupts?

1. Defeats the purpose of multiprogramming.
2. Programmer error may result in them never being re-enabled (e.g., due to a logic error, or program crash)

What is a semaphore?

A semaphore is a synchronization primitive provided by an OS. It consists of a shared variable, two atomic operations, and a process queue.

What is the difference between a counting semaphore and a binary semaphore?

A counting semaphore initialized to N allows N processes to access the shared resource. A binary semaphore is a counting semaphore with N = 1.

Describe the two semaphore operations, wait and post:

	<u>Binary</u>	<u>Counting</u>
wait(s)	if(s == 1) s = 0; else block();	s--; if(s < 0) block();
post(s)	if(none blocked) s = 1; else wakeup();	s++; if(s <= 0) wakeup();
		s < 0 ⇒ s == # processes blocked on s

If you just want mutual exclusion, will a binary semaphore suffice?

Yes: the 'shared resource' is 'entry to the critical section'.
A binary semaphore will allow only one process to access the shared resource, i.e., enter the critical section.

Show how a semaphore can be used to synchronize two processes, such that P0 executes statement S0 before P1 executes statement S1:

```
        Binary semaphore s = 0;
P0      P1
.       .
.       .
S0      wait(s) <- P1 blocks until s = 1
P0 sets s to 1 -> post(s)      S1
.       .
.       .
```

What is an exit handler?

A function to be called by a user process when it terminates, typically to do some final 'cleanup' (e.g., de-allocate memory, ...)
It is registered using `atexit()`.
Exit handlers are called in the reverse order of registration.

What is a signal handler?

A function to be called by a user process on receipt of a signal.
It is registered using `signal()`.

Describe the effect of `dup(fd)`

File descriptor `fd` is duplicated on the lowest available descriptor.

Describe the effect of `dup2(fd, onto)`

'onto' is closed. `fd` is duplicated on 'onto'.

Describe the effect of `pipe(fd)`

A one-way 'communication channel' is opened, such that `fd[1]` can be written to, and `fd[0]` can be read from.