Inter-Process Communication

Signals

A signal is a software equivalent of a hardware interrupt: they can be sent to a process by the kernel or by a process

Inter-Process Communication

Signals

A signal is a software equivalent of a hardware interrupt: they can be sent to a process by the kernel or by a process

Also: *raised* and *initiated*

Inter-Process Communication

Signals

A signal is a software equivalent of a hardware interrupt: they can be sent to a process by the kernel or by a process

Also: *raised* and *initiated*

Just like a hardware interrupt, when a process receives a signal, it stops what it currently doing and goes off to execute a *signal handler*, in direct analogy with an interrupt handler

Inter-Process Communication

Signals

A signal is a software equivalent of a hardware interrupt: they can be sent to a process by the kernel or by a process

Also: *raised* and *initiated*

Just like a hardware interrupt, when a process receives a signal, it stops what it currently doing and goes off to execute a *signal handler*, in direct analogy with an interrupt handler

Handled within the user program: the signal handler is just some code in the program, written by the programmer

Inter-Process Communication

Signals

Again, what we lazily say is not quite what really happens

Inter-Process Communication

Signals

Again, what we lazily say is not quite what really happens

When a signal is raised (which needs a syscall) the OS takes over and notes the signal in the receiving process' PCB

Inter-Process Communication

Signals

Again, what we lazily say is not quite what really happens

When a signal is raised (which needs a syscall) the OS takes over and notes the signal in the receiving process' PCB

When the OS next runs that process, it jumps to the signal handler code within the process, rather than to the place where the process was preempted

Inter-Process Communication

Signals

A process can send a signal to another process (or even itself) that has the same owner (same userid)

Inter-Process Communication

Signals

A process can send a signal to another process (or even itself) that has the same owner (same userid)

The normal restrictions on userids applies: only root can send a signal to another user's process

Inter-Process Communication

Signals

A process can send a signal to another process (or even itself) that has the same owner (same userid)

The normal restrictions on userids applies: only root can send a signal to another user's process

But remember that all signals are delivered to processes via the kernel

Inter-Process Communication

Signals

A process can send a signal to another process (or even itself) that has the same owner (same userid)

The normal restrictions on userids applies: only root can send a signal to another user's process

But remember that all signals are delivered to processes via the kernel

The kernel can also itself initiate a signal, e.g., a signal to indicate activity of a peripheral

Inter-Process Communication

Signals

A process can send a signal to another process (or even itself) that has the same owner (same userid)

The normal restrictions on userids applies: only root can send a signal to another user's process

But remember that all signals are delivered to processes via the kernel

The kernel can also itself initiate a signal, e.g., a signal to indicate activity of a peripheral

Naturally, the kernel can send signals to any process

Inter-Process Communication

Signals

Use the POSIX function `kill()` to send a signal in a user program

Inter-Process Communication

Signals

Use the POSIX function `kill()` to send a signal in a user program

And functions like `signal()`, `sigaction()`, `sigaddset()` and more to manage signals

Inter-Process Communication

Signals

A signal is in essence a flag (a single bit), but there are many different types of signal, e.g., HUP, INT, SEGV, KILL, PIPE, etc., to indicate different kinds of events

Inter-Process Communication

Signals

A signal is in essence a flag (a single bit), but there are many different types of signal, e.g., HUP, INT, SEGV, KILL, PIPE, etc., to indicate different kinds of events

A process can, to some extent, choose how to react to a signal

Inter-Process Communication

Signals

A signal is in essence a flag (a single bit), but there are many different types of signal, e.g., HUP, INT, SEGV, KILL, PIPE, etc., to indicate different kinds of events

A process can, to some extent, choose how to react to a signal

- ignore it: that is, inform the kernel that it does not want to receive a particular signal, so the kernel will not note delivery in the PCB as above

Inter-Process Communication

Signals

A signal is in essence a flag (a single bit), but there are many different types of signal, e.g., HUP, INT, SEGV, KILL, PIPE, etc., to indicate different kinds of events

A process can, to some extent, choose how to react to a signal

- ignore it: that is, inform the kernel that it does not want to receive a particular signal, so the kernel will not note delivery in the PCB as above
- accept it and act on it: run the signal handler code

Inter-Process Communication

Signals

A signal is in essence a flag (a single bit), but there are many different types of signal, e.g., HUP, INT, SEGV, KILL, PIPE, etc., to indicate different kinds of events

A process can, to some extent, choose how to react to a signal

- ignore it: that is, inform the kernel that it does not want to receive a particular signal, so the kernel will not note delivery in the PCB as above
- accept it and act on it: run the signal handler code
- suspend (voluntarily relinquish)

Inter-Process Communication

Signals

A signal is in essence a flag (a single bit), but there are many different types of signal, e.g., HUP, INT, SEGV, KILL, PIPE, etc., to indicate different kinds of events

A process can, to some extent, choose how to react to a signal

- ignore it: that is, inform the kernel that it does not want to receive a particular signal, so the kernel will not note delivery in the PCB as above
- accept it and act on it: run the signal handler code
- suspend (voluntarily relinquish)
- terminate

Inter-Process Communication

Signals

Some signals cannot be ignored or otherwise acted upon, in particular a KILL signal, which always terminates the process (i.e., that process will be moved to the exit state)

Inter-Process Communication

Signals

Some signals cannot be ignored or otherwise acted upon, in particular a KILL signal, which always terminates the process (i.e., that process will be moved to the exit state)

This is why signals are regulated by the kernel; a user can kill their own processes, but not others'

Inter-Process Communication

Signals

Some signals cannot be ignored or otherwise acted upon, in particular a KILL signal, which always terminates the process (i.e., that process will be moved to the exit state)

This is why signals are regulated by the kernel; a user can kill their own processes, but not others'

And root (administrator) can kill any process

Inter-Process Communication

Signals

Some signals cannot be ignored or otherwise acted upon, in particular a KILL signal, which always terminates the process (i.e., that process will be moved to the exit state)

This is why signals are regulated by the kernel; a user can kill their own processes, but not others'

And root (administrator) can kill any process

Signals are *asynchronous*, that is they might arrive at any point during the running of the program

Inter-Process Communication

Signals

Some signals cannot be ignored or otherwise acted upon, in particular a KILL signal, which always terminates the process (i.e., that process will be moved to the exit state)

This is why signals are regulated by the kernel; a user can kill their own processes, but not others'

And root (administrator) can kill any process

Signals are *asynchronous*, that is they might arrive at any point during the running of the program

Programs that use signals must be written accordingly

Inter-Process Communication

Signals

Some signals cannot be ignored or otherwise acted upon, in particular a KILL signal, which always terminates the process (i.e., that process will be moved to the exit state)

This is why signals are regulated by the kernel; a user can kill their own processes, but not others'

And root (administrator) can kill any process

Signals are *asynchronous*, that is they might arrive at any point during the running of the program

Programs that use signals must be written accordingly

(And they will always arrive at the most inconvenient time. . .)

Inter-Process Communication

Signals

There exist default signal actions for each type of signal, but a program must include its own handler functions if it wants to do something other than the default action when it receives a signal

Inter-Process Communication

Signals

There exist default signal actions for each type of signal, but a program must include its own handler functions if it wants to do something other than the default action when it receives a signal

When a process receives a signal, it stops what it is doing, saves its state and calls the signal handler

Inter-Process Communication

Signals

There exist default signal actions for each type of signal, but a program must include its own handler functions if it wants to do something other than the default action when it receives a signal

When a process receives a signal, it stops what it is doing, saves its state and calls the signal handler

Exercise. Rewrite the above sentence to describe what really happens. (No program you have ever written includes code to save its state!)

Inter-Process Communication

Signals

There exist default signal actions for each type of signal, but a program must include its own handler functions if it wants to do something other than the default action when it receives a signal

When a process receives a signal, it stops what it is doing, saves its state and calls the signal handler

Exercise. Rewrite the above sentence to describe what really happens. (No program you have ever written includes code to save its state!)

So this is very analogous to an OS interrupt

Inter-Process Communication

Signals

There exist default signal actions for each type of signal, but a program must include its own handler functions if it wants to do something other than the default action when it receives a signal

When a process receives a signal, it stops what it is doing, saves its state and calls the signal handler

Exercise. Rewrite the above sentence to describe what really happens. (No program you have ever written includes code to save its state!)

So this is very analogous to an OS interrupt

If and when the handler code finishes, the process continues from where it was interrupted

Inter-Process Communication

Signals

Example signals

```
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS      34) SIGRTMIN
35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4
39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Inter-Process Communication

Signals

- INT: a general interrupt

Inter-Process Communication

Signals

- INT: a general interrupt
- ILL: sent by the kernel to a process when it has tried to use a privileged or non-existent machine instruction

Inter-Process Communication

Signals

- INT: a general interrupt
- ILL: sent by the kernel to a process when it has tried to use a privileged or non-existent machine instruction
- KILL: non-ignorable terminate

Inter-Process Communication

Signals

- INT: a general interrupt
- ILL: sent by the kernel to a process when it has tried to use a privileged or non-existent machine instruction
- KILL: non-ignorable terminate
- SEGV: sent by the kernel to a process when it has tried to access memory it shouldn't

Inter-Process Communication

Signals

- INT: a general interrupt
- ILL: sent by the kernel to a process when it has tried to use a privileged or non-existent machine instruction
- KILL: non-ignorable terminate
- SEGV: sent by the kernel to a process when it has tried to access memory it shouldn't
- ALRM: a timer signal (*not* the preemption timer!)

Inter-Process Communication

Signals

- INT: a general interrupt
- ILL: sent by the kernel to a process when it has tried to use a privileged or non-existent machine instruction
- KILL: non-ignorable terminate
- SEGV: sent by the kernel to a process when it has tried to access memory it shouldn't
- ALRM: a timer signal (*not* the preemption timer!)
- USR1 and USR2: signals for the use of user programs

Inter-Process Communication

Signals

- INT: a general interrupt
- ILL: sent by the kernel to a process when it has tried to use a privileged or non-existent machine instruction
- KILL: non-ignorable terminate
- SEGV: sent by the kernel to a process when it has tried to access memory it shouldn't
- ALRM: a timer signal (*not* the preemption timer!)
- USR1 and USR2: signals for the use of user programs
- RT: a large number of signals are provided for real-time processing

Inter-Process Communication

Signals

- INT: a general interrupt
- ILL: sent by the kernel to a process when it has tried to use a privileged or non-existent machine instruction
- KILL: non-ignorable terminate
- SEGV: sent by the kernel to a process when it has tried to access memory it shouldn't
- ALRM: a timer signal (*not* the preemption timer!)
- USR1 and USR2: signals for the use of user programs
- RT: a large number of signals are provided for real-time processing
- Signals 32 and 33 are not used in the OS in this example

Inter-Process Communication

Signals

Each signal has a default action

Inter-Process Communication

Signals

Each signal has a default action

- INT, ILL, ALRM, SEGV, USR1, USR2: exit the program

Inter-Process Communication

Signals

Each signal has a default action

- INT, ILL, ALRM, SEGV, USR1, USR2: exit the program
- TSTP: suspend

Inter-Process Communication

Signals

Each signal has a default action

- INT, ILL, ALRM, SEGV, USR1, USR2: exit the program
- TSTP: suspend
- CONT: continue after a TSTP

Inter-Process Communication

Signals

Each signal has a default action

- INT, ILL, ALRM, SEGV, USR1, USR2: exit the program
- TSTP: suspend
- CONT: continue after a TSTP
- CHLD: ignore

Inter-Process Communication

Signals

Each signal has a default action

- INT, ILL, ALRM, SEGV, USR1, USR2: exit the program
- TSTP: suspend
- CONT: continue after a TSTP
- CHLD: ignore

Most default actions can be overridden by the program, some, notably KILL, cannot

Inter-Process Communication

Signals

Signals are

Inter-Process Communication

Signals

Signals are

- Fast and efficient

Inter-Process Communication

Signals

Signals are

- Fast and efficient
- Asynchronous

Inter-Process Communication

Signals

Signals are

- Fast and efficient
- Asynchronous
- Used a very great deal

Inter-Process Communication

Signals

Signals are

- Fast and efficient
- Asynchronous
- Used a very great deal
- Only transmit a small amount of information

Inter-Process Communication

Signals

Signals are

- Fast and efficient
- Asynchronous
- Used a very great deal
- Only transmit a small amount of information
- So often are used in concert with other IPC mechanisms

Inter-Process Communication

Signals

Signals are

- Fast and efficient
- Asynchronous
- Used a very great deal
- Only transmit a small amount of information
- So often are used in concert with other IPC mechanisms
- Are a bit fiddly to program correctly