PennState

CMPSC 311 - Introduction to Systems Programming

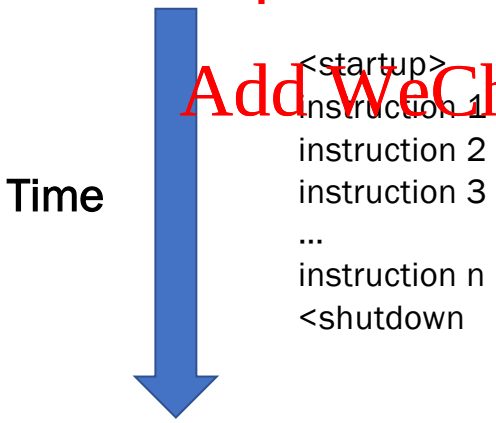
Signals

Professors:
Suman Saha
(Slides are mostly by *Professor Patrick McDaniel* and *Professor Abutalib Aghayev*)

Control Flow

PennState

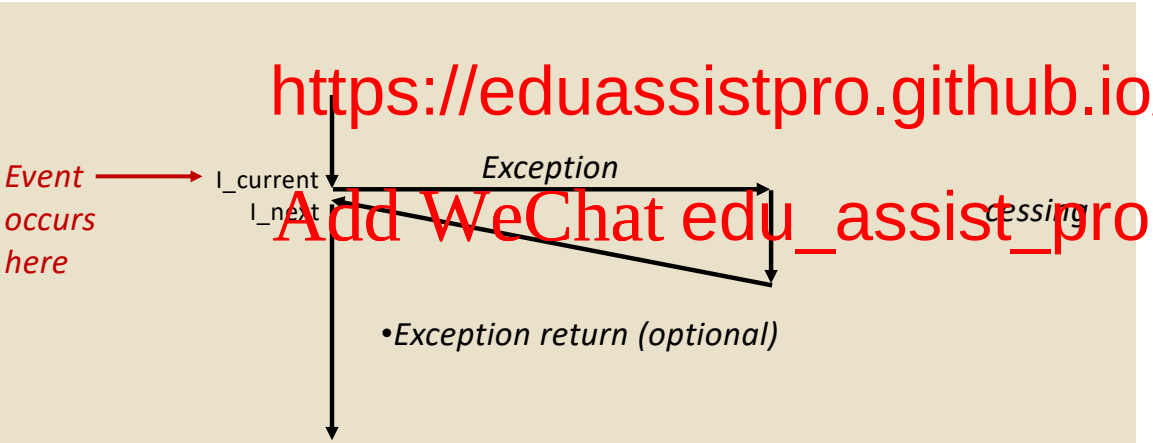
- Processors do only one thing:
  - from startup to shutdown, a CPU simply reads and executes a sequence of instructions, one at a time
  - This sequence is the C l)



Exceptional Control Flow

PennState

- Exceptional control flow enables a system to react to an event



Exceptional Control Flow

PennState

- Mechanisms exists at all levels of a computer system for exceptional control
- Low-level mechanisms
  - Exceptions
    - Examples: interrupt
  - Implemented using co ftware
- High-level mechanisms
  - Process context switch (implemented by OS ardware timer)
  - Signals (implemented by OS software)
  - Nonlocal jumps: setjmp() and longjmp() (implemented by C runtime library)

UNIX Signals

PennState

- A signal is a special message sent through the OS to tell a process (or thread) of some command or event
- The process execution "signal handler" code runs.
- The process can resume operation after the signal handling is complete.

Signal types (abbreviated)

PennState

```
/* Signals */
#define SIGHUP          1       /* Hangup (POSIX).  */
#define SIGINT          2       /* Interrupt (ANSI).  */
#define SIGQUIT
#define SIGABRT
#define SIGFPE                  ption (ANSI).  */
#define SIGKILL         9       /* Kill, unblockable (POSIX).  */
#define SIGSEGV         11      /* Segme                  (ANSI).  */
#define SIGTERM         15      /* Termi                  */
#define SIGSTKFLT       16      /* Stack
#define SIGCHLD         17      /* Child status has changed (POSIX).  */
#define SIGCONT         18      /* Continue (POSIX).  */
#define SIGSYS          31      /* Bad system call.  */
```

Signals as process control

PennState

- The operating system use signals to control process behavior
  - Signals are sent on errors

```
#define SIGILL          4       /* Illegal instruction (ANSI).  */
#define SIGTRAP         5       /* Trace trap (POSIX).  */
#define SIGIOT
#define SIGBUS
#define SIGFPE                  eption (ANSI).  */
#define SIGSEGV         11      /* Segm                 n (ANSI).  */
```

  - Signals can be used by other applications to

```
#define SIGUSR1         10      /* User                  (POSIX).  */
#define SIGUSR2         12      /* User-defined signal 2 (POSIX).  */
```

  - Control the process execution

```
#define SIGKILL         9       /* Kill, unblockable (POSIX).  */
#define SIGCONT         18      /* Continue (POSIX).  */
#define SIGSTOP         19      /* Stop, unblockable (POSIX).  */
```

Process IDs

PennState

- Every process running on the OS is given a unique process ID (PID)
  - This is what is used in the OS and for process control to reference tha
    program instance.
- To find a process ID for a program, use the ps utility
  - The ps stands for “process status”

```
$ ps -U mcdaniel
  PID TTY          TIME CMD
30908 ?        00:00:00 gnome-keyring-d
30919 ?        00:00:00 gnome-session
30964 ?        00:00:00 ssh-agent
30967 ?        00:00:00 dbus-launch
30968 ?        00:00:01 dbus-daemon
      ?        00:00:00 at-spi-bus-laun
      ?        00:00:00 dbus-daemon
      ?        00:00:00 at-spi2-registr
      ?        00:00:02 gnome-settings-
31009 ?        00:00:00 pulseaudio
               00:00:00 gvfsd
               00:00:00 gvfsd-fuse
               00:02:43 compiz
               00:00:00 dconf-service
31044 ?        00:00:00 gnome-fallback-
31045 ?        00:00:06 nautilus
31047 ?        00:00:01 nm-applet
31048 ?        00:00:41 vmtoolsd
31049 ?        00:00:00 polkit-gnome-au
31064 ?        00:00:00 gvfs-udisks2-vo
31079 ?        00:00:00 gvfs-gphoto2-vo
31083 ?        00:00:00 gvfs-afc-volume
31090 ?        00:00:00 gvfs-mtp-volume
...
```

kill

PennState

- Kill is a program than sends signals to processes.

```
kill [-<sig>] <pid>
```


- Where `<sig>` is the signal number and `<pid>` is the process ID of the running program you w
  - If no `SIGNUM` is given, .


```
$ ps -U mcdaniel
57613 pts/4    00:00:00 signals
$ kill -1 57613
$ kill -2 57613
$ kill -9 57613
```

```
$
Sl
Signal handler got a SIGHUP!
Signals received : 1
Woken up!!
Sleeping ...zzzzz ....
Signal handler got a SIGNINT!
Signals received : 2
Woken up!!
Sleeping ...zzzzz ....
Killed
```

SIGTERM vs. SIGKILL

PennState

- `SIGTERM` interrupts the program and asks it to shut down, which it should.
  - Sometimes this does not work (for instance when the process is in a locked state)
  - It is often desirable to add a signal handler to handle the SIGTERM, so that it can gracefully shut down th files, etc.
- `SIGKILL` kills the proc
  - Can lead to inconsistent state, because ther ity to gracefully shutdown the process.

Definition: the term *graceful shutdown* refers to the proper and complete sync with secondary storage, disposal of resources, and normal termination.

killall

PennState

- Killall is a program than sends signals to all instances of a particular progam.

`killall [-<sig>] <name>`

- Where `<sig>` is the signal number and `<name>` is the name of running program you want to s
  - If no `SIGNUM` is given,


```
$ killall -1 signals
$ killall -2 signals
$ killall -SIGKILL signals
```

```
$ .
Sleeping ...zzzzz ....
Signal handler got a SIGHUP!
Signals received : 1
Woken up!!
Sleeping ...zzzzz ....
Signal handler got a SIGNINT!
Signals received : 2
Woken up!!
Sleeping ...zzzzz ....
Killed
```

raise()

PennState

- raise allows a process to send signals to itself.

```
int raise(int sig);
```


- There are a range of reasons why a process might want to do this.
  - Suspend itself (SIGST
  - Kill itself (SIGKILL)
  - Reset its configuation (SIGHUP)
  - User defined signals (SIGUSR1..)


```c
void suicide_signal(void) {
    raise(SIGKILL);
    return; // This will never be reached
}
```

User-defined signal handlers

PennState

- You can create your own signal handlers simply by creating a function

```
void <fname>( int <var_name> )
```


- and passing a function pointer to the function

```
r_t handler(int);
sig
```


- Thereafter, whenever your program is called instead of the default handler.


```
void signal_handler(int no) {
    printf("Sig handler got a [%d]\n", no);
    return;
}
```

```
    signal(SIGHUP, signal_handler);
    signal(SIGINT, signal_handler);
```

Function pointers

PennState

- A function pointer is a pointer to a function that can be assigned, passed as parameters, and called


```
<return> (*<var>)(<params>);
```


- `<return>` is the retur
- `<var>` is the variable
- `<params >` are the parameters , separate


```
int myfunc(int i) {
  printf("Got into function with %d\n", i);
  return 0;
}

int main( void ) {
  int (*func)(int);
  func = myfunc;
  func(7);
  return 0;
}
```

```
$ ./signals
Got into function with 7
$
```

An alternate approach

PennState

- The `sigaction()` system call changes the action taken by a process on receipt of a specific signal.

```
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
```


- Where:

  - `signnum` - is the signal number to be handled
  - `act` - is a structure containing information about the new h ignore the signal
  - `oldact` - is a pointer to the previously assigned handler, a unction


```
struct sigaction new_action, old_action;
new_action.sa_handler = signal_handler;
new_action.sa_flags = SA_NODEFER | SA_ONSTACK;
sigaction(SIGINT, &new_action, &old_action);
```

Why another API?

PennState

- Many argue that the `sigaction` function is better:
  - The signal() function does not block other signals from arriving while the current handler is executing; sigaction() can block other signals until the current handler returns.
  - The signal() function resets the signal action back to SIG_DFL (default) for almost all signals.
  - Better tuning of signals/contr
    - SA_NODEFER - don't susp
    - SA_ONSTACK - provide alternate stack for signal han
    - SA_RESETHAND - Restore the signal action to the defaul nal handler.

Note: *In general*, `sigaction` is preferred over signal.

Putting it all together ...

PennState

```c
void signal_handler(int no) {
    printf("Signal received : %d\n", no);
    if (no == SIGHUP) {
        printf("Signal handler got a SIGHUP!\n");
    } else if (no == SIGINT) {
        printf("Signal handler got a SIGNINT!\n");
    }
    return;
}
void cleanup_handler(int
    printf("Killed");
    exit(0);
}
int main(void) {
    struct sigaction new_action, old_action; // Setup
    new_action.sa_handler = signal_handler;
    new_action.sa_flags = SA_NODEFER | SA_ONSTACK;
    sigaction( SIGINT, &new_action, &old_action );
    signal( SIGHUP, signal_handler );      // Setup the signal handlers
    signal( SIGTERM, cleanup_handler );

    while (1) {
        printf( "Sleeping ...zzzzz ....\n" );
        select( 0, NULL, NULL, NULL, NULL );
        printf( "Woken up!!\n" );
    }

    // Return successfully
    return 0;
}
```

```
$ ./signals
Sleeping ...zzzzz ....
Signal received : 1
Signal handler got a SIGHUP!
Woken up!
Sleeping ...zzzzz ....
Signal received : 2
dler got a SIGNINT!
```