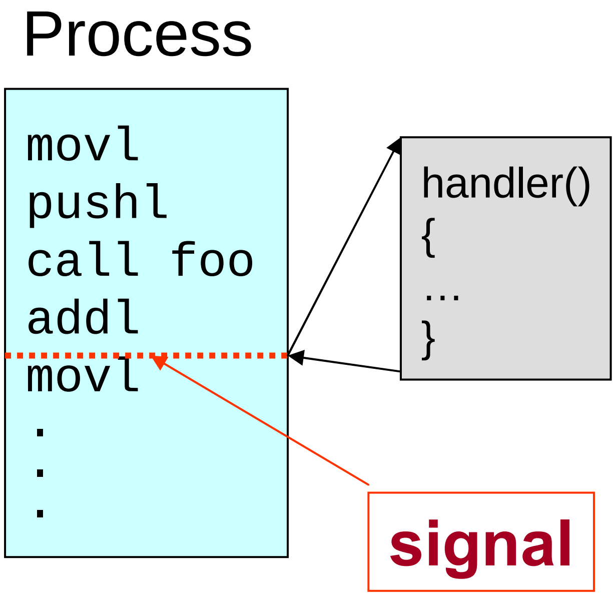
Signals and
Writing Portable Programs and
Course Wrap-Up

Prof. David August

COS 217

Signals

- Event notification sent to a process at *any* time
  - An event generates a signal
  - OS stops the process immediately
  - Signal handler executes and completes
  - The process resumes where it left off



Process

```
movl
pushl
call foo
addl
movl
.
.
.
```

```
handler()
{
…
}
```

signal

Signals Can Originate From:

- Keyboard:
  - Ctrl-C → INT signal (process terminates)
  - Ctrl-Z → TSTP signal (process suspe
  - Ctrl-\ → ABRT signal (process dumps e)

- Program itself:
  - Illegal memory reference → SIGSEGV (segmentation fault)
  - The **`kill`** and **`raise`** library functions. Example: send a signal to self:

```
if (kill(getpid(), SIGABRT))
   exit(0);
```

Signals Can Originate From:

- Command Line:

`kill -<signal> <PID>`

- Example: **`kill -INT 1234`**
  - Send the INT signal to process with PID 1234
  - Same as pressing Ctrl-C if process 1234 is running
- If no signal specified, the default is SIGTERM

`fg` (foreground)

- On UNIX shells, this command sends a **`CONT`** signal
- Resume execution of the process (that was suspended with Ctrl-Z or a command "**`bg`**")
- See man pages for **`fg`** and **`bg`**

Predefined and Defined Signals

- Find out the predefined signals

```
% kill –l
% HUP INT QUIT ILL TRAP ABRT BUS FPE KILL
  USR1 SEGV USR2 PIPE ALRM TERM STKFLT CHLD
  CONT STOP TSTP TTIN TTOU URG XCPU XFSZ
  VTALRM PROF WINCH POLL PWR SYS RTMIN
  RTMIN+1 RTMIN+2 RTMIN+3 RTMAX-3 RTMAX-2
  RTMAX-1 RTMAX
```

- Applications can define their own signals
  - An application can define signals with unused values

Some Predefined Signals in UNIX

```
#define SIGHUP          1       /* Hangup (POSIX).  */
#define SIGINT          2       /* Interrupt (ANSI).  */
#define SIGQUIT         3       /* Quit (POSIX).  */
#define SIGILL          4       /* Illegal instruction (ANSI).  */
#define SIGTRAP         5       /* Trace trap (POSIX).  */
#define SIGABRT         6       /* Abort (ANSI).  */
#define SIGFPE          8       /* Floating-point exception (ANSI).  */
#define SIGKILL         9       /* Kill, unblockable (POSIX).  */
#define SIGUSR1         10      /* User-defined signal 1 (POSIX).  */
#define SIGSEGV         11      /* Segmentation violation (ANSI).  */
#define SIGUSR2         12      /* User-defined signal 2 (POSIX).  */
#define SIGPIPE         13      /* Broken pipe (POSIX).  */
#define SIGALRM         14      /* Alarm clock (POSIX).  */
#define SIGTERM         15      /* Termination (ANSI).  */
#define SIGCHLD         17      /* Child status has changed (POSIX).  */
#define SIGCONT         18      /* Continue (POSIX).  */
#define SIGSTOP         19      /* Stop, unblockable (POSIX).  */
#define SIGTSTP         20      /* Keyboard stop (POSIX).  */
#define SIGTTIN         21      /* Background read from tty (POSIX).  */
#define SIGTTOU         22      /* Background write to tty (POSIX).  */
#define SIGPROF         27      /* Profiling alarm clock (4.2 BSD).  */
```

Signal Handling

- Signals have default handlers
  - Usually, terminate the process and generate core image
- Programs can over-ride default for most signals
  - Define their own handlers
  - Ignore certain signals, or temporarily block them
- Two signals are not “catchable” in user programs
  - KILL
    - Terminate the process immediately
    - Catchable termination signal is TERM
  - STOP
    - Suspend the process immediately
    - Can resume the process with signal CONT
    - Catchable suspension signal is TSTP

Installing A Signal Handler

- Predefined signal handlers
  - **SIG_DFL:** Default handler
  - **SIG_IGN:** Ignore the signal
- To install a handler, use

```
#include <signal.h>

typedef void (*sighandler_t)(int);
sighandler_t signal(int sig, sighandler_t handler);
```

  - Handler will be invoked, when signal **sig** occurs
  - Return the old handler on success; **SIG_ERR** on error
  - On most UNIX systems, after the handler executes, the OS resets the handler to **SIG_DFL**

Example: Clean Up Temporary File

- Program generates a lot of intermediate results
  - Store the data in a temporary file (e.g., “temp.xxx”)
  - Remove the file when the program ends (i.e., unlink)

```
#include <stdio.h>

char *tmpfile = “temp.xxx”;
int main() {
   FILE *fp;
   fp = fopen(tmpfile, “rw”);
   …
   fclose(fp);
   unlink(tmpfile);
   return(0);
}
```

Solution: Clean-Up Signal Handler

```
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
char *tmpfile = “temp.xxx”;

void cleanup(void) {
   unlink(tmpfile);
   exit(EXIT_FAILURE);
}
int main(void) {
   if (signal(SIGINT, cleanup) == SIG_ERR)
      fprintf(stderr, “Cannot set up signal\n”);
  …
   return(0);
}
```

Portability

- Multiple kinds of hardware
  - 32-bit Intel Architecture
  - 64-bit IA, PowerPC, Sparc, MIPS, Arms, …
- Multiple operating systems
  - Linux
  - Windows, Mac, Sun, AIX, …
- Multiple character sets
  - ASCII
  - Latin-1, unicode, …
- Multiple byte orderings
  - Little endian
  - Big endian

Size of Data Types

- What are the sizes of **`char, short, int, long, float and double`** in C and C++?
  - **`char`** has at least 8 bits, **`short`** and **`int`** at least 16 bits
  - **`sizeof(char)`** ≤ **`sizeof(short)`** ≤ **`sizeof(int)`** ≤ **`sizeof(long)`**
  - **`sizeof(float)`** ≤ **`sizeof(double)`**
- In Java, sizes are defined
  - **`byte`**: 8 bits
  - **`char`**: 16 bits
  - **`short`**: 16 bits
  - **`int`**: 32 bits
  - **`long`**: 64 bits
- **Our advice: always use `sizeof()` to be safe**

Order of Evaluation

- Order of evaluation may be ambiguous
  - `strings[i] = names[++i];`
    - `i` can be incremented before or after indexing `strings`!
  - `printf(“%c %c\n”, getchar(), getchar());`
    - The second character in `stdin` can be printed first!
- What are the rules in C and C++?
  - Side effects and function calls must be completed at “**;**”
- **Our advice: do not depend on the order of evaluation in an expression**

Alignment of Structures and Unions

- Structure consisting of multiple elements

```
struct Foo {
    char x;
    int y;
}
```

- Items are laid out in the order of declaration
- But, the alignment is undefined
  - There might be holes between the elements
  - E.g., **y** may be 2, 4, or 8 bytes from **x**

Internationalization

- Don’t assume ASCII
  - Many countries do not use English
  - Asian languages use 16 bits per character
- Standardizations
  - Latin-1 augments ASCII by using all 8 bits
  - Unicode uses 16 bits per character
  - Java uses Unicode as its native character set for strings
- Issues with Unicode
  - Byte order issue!
  - Solution: use UTF-8 as an intermediate representation or define the byte order for each character

Avoid Conditional Compilation

- Writing platform-specific code is possible

```
…
some common code
#ifdef MAC
…
#else
#ifdef WINDOWSXP
…
#endif
#endif
```

- But, #ifdef code is difficult to manage
  - Platform-specific code may be all over the place
  - Plus, each part requires separate testing

Isolation

- Common feature may not always work: Life is hard
- Localize system dependencies in separate files
  - Separate file to wrap the interface calls for each system
  - Example: unix.c, windows.c, mac.c, …
- Hide system dependencies behind interfaces
  - Abstraction can serve as the boundary between portable and non-portable components
- Java goes one big step further
  - Virtual machine which abstracts the entire machine
  - Independent of operating systems and the hardware

Course Wrap Up

Lessons About Computer Science

- Modularity
  - Well-defined interfaces between components
  - Allows changing the implementation of one component without changing another
  - The key to managing complexity in large systems
- Resource sharing
  - Time sharing of the CPU by multiple processes
  - Sharing of the physical memory by multiple processes
- Indirection
  - Representing address space with virtual memory
  - Manipulating data via pointers (or addresses)

Lessons Continued

- Hierarchy
  - Memory: registers, cache, main memory, disk, tape, …
  - Balancing the trade-off between fast/small and slow/big
- Bits can mean anything
  - Code, addresses, characters, pixels, money, grades, …
  - Arithmetic is just a lot of logic operations
  - The meaning of the bits depends entirely on how they are accessed, used, and manipulated

Stay tuned for final exam review session details…