Announcements

- Program #2 due today
- Midterm #2 is next Tuesday
  - same time and place as last time

Command Line Arguments

- int main(int argc, char *argv[])
  - argc - number of arguments (including program name)
  - argv - array of command line arguments
    - argv[0] - command invoked to start program
    - argv[n] - n'th command line argument
- Example:
  - ./foo -file myfile -help
  - argv[0] = "./foo"
  - argv[1] = "-file"
  - argv[2] = "myfile"
  - argv[3] = "-help"

Programming Command Line Arguments

- Generally, want arguments to be accepted in any order
  - ./foo -file myfile -help
  - ./foo -help -file myfile
- Solution:
  - Table driven command line parsing
    - Example of a common pattern, table drive parsing
  - Declare a structure
    - fields describe each valid option

```
typedef enum { Flag, StrParam, IntParam };
typedef struct {
    char *name;
    optionTypes options;
    void *parameter;
} option;
```

Command Line Arguments Continued

```
int debugFlag, limit;
char *inputFileName;
optionTable = [] {
        { "-debug", Flag, &debugFlag },
        { "-file", StrParam, &inputFileName },
        { "-limit", IntParam, &limit },
        …. /* other options go here */
 };

int optionCount = sizeof(optionTable)/sizeof(option);
```

Project 1B Using Table Driven Parsing

```
typedef struct {
    char *name;
    int opCode;
    int numRegisters;
    int usesMem;
} opInfo;
```

```
static opInfo opTable[] = {
    { "Load",   1, 1, 1 },
    { "Move",   2, 2, 0 },
    { "Store",  3, 1, 1 },
    { "Add",    4, 3, 0 },
    { "Halt",   5, 0, 0 },
    { "Negate", 6, 1, 0 },
    { "Branch", 7, 2, 1 },
    { "Bnn",    8, 1, 1 },
    { "Input",  10, 1, 0 },
    { "Output", 11, 1, 0 },
    { "Data",   -1, 0, 1 },
};
```

Environment Variables

- Keyword, argument pairs
- Allow passing info into programs
- Examples:
  - PATH - used by shell to find programs
  - PRINTER - default printer to use
- Passed as the third parameter to main
  - int main(int argc, char *argv[], char *envp[])
  - consists of a null terminated array of characters
    - KEYWORD=VALUE stored in each one
- C helper functions:
  - char *getenv(const char *keyword);
  - char *putenv(const char *string);
    - where string is KEYWORD=VALUE

Environment Variable Example

- Shell Commands to Setup Environement:
  - C Shell: setenv FOO_CONFIG ~/foo
  - Bash: FOO_CONFIG=~/foo

```
main(int argc, char *argv[], char *envp[])
{
    char *configDirectory;

    configDirectory = getenv("FOO_CONFIG");

}
```

Creating New Processes

- Use fork system call
  - UNIX (and LINUX) specific
  - creates a new copy of the current process
- Syntax int fork();
  - returns twice
    - once from initial call
    - second time in a new process
  - return value indicates which process is which
    - 0 the "child" (new) process
    - > 0 the "parent" (original) process
    - < 0 the "parent", but not child was created (error case)

Fork Example

```
int main(int argc, char *argv[]) {
        int pid;

        pid = fork();
        if (pid == 0) {
                printf("I am the child\n");
            exit(0);
        } else if (pid > 0) {
            printf("Created child process %d\n", pid);
            exit(0);
        } else {
            printf("Error, unable to create a new process);
            exit(0);
        }
}
```

Learning About Other Processes

- Wait and waitpid system call
  - #include <sys/types.h>
  - #include <sys/wait.h>
  - pid_t wait(int *status);
    - wait until a child process terminates
  - pid_t waitpid(pid_t pid, int *status, int options);
    - wait until the passed process terminates
  - return is the id the the terminated process
  - status fills out a status variable
    - WIFEXITED(status) - true if the child terminated normally
    - WEXITSTATUS(status) - low 8 bits of parameter to exit
    - WTERMSIG(status) - signal number that terminated process

Invoking a new program

- exec system calls
  - int execv(const char *prog, char *const argv[]);
    - run the program in the passed prog parameter
    - pass the new program argv
  - int execvp(const char *file, char *const argv[]);
    - like execv, but used the PATH environment variable
- On success,
  - new program launched, the system call does not return
- On failure,
  - returns -1, sets global errno to indicate cause of the error

Fork and Exec Example

```
    char *args[] = { "ls", "-l", "-a", NULL };
    ….
    pid = fork();
    if (pid == 0) {
        ret = execvp("ls", args);
        if (ret) {
            perror("execvp");
            exit(-1);
        }
    } else if (pid > 0) {  …
    } else {  …
    }
    …..
}
```

Additional I/O system Calls

- Sometimes process want to communicate
  - abstraction: pipe - one process writes, other reads
  - shell example: ls | wc
- int pipe(int filedes[2]);
  - create a pipe between two processes
  - write and read system call can use these
- Stndard I/O
  - fd 0 is standard input to a process
  - fd 1 is standard output from a process
  - fd 2 is standard error output from
- int dup2(int oldfd, int newfd);
  - change output of one fd to another
  - dup2(0, myfd)
    - now all standard input comes from myfd

Changing Standard Input/Output

- Useful to combine pipe, dup2, and exec

```
….
pipe(pipefds);
pid = fork();
if (pid == 0) {
      dup2(0, pipefd[0]);
      close(pipefd[1]);
      ret = execvp("ls", args);
      ….
else if (pid > 0) {
      dup2(1, pipefd[1]);
      close(pipefd[0]);
} else {
   ….
}
```

A Simple Shell Program

```
while (1)  {
        ret = fgets(line, sizeof(line), stdin);
        …
        /* convert line into array of arguments */
        pid = fork();
        if (pid == 0) {
                ret = execvp(args[0], args);
            if (ret) {
                 printf("Command not found\n");
                 exit(-1);
            } else if (pid > 0) {
                ret = waitpid(pid, &status, NULL);
            } else { … }
}
```