Computer Systems Fundamentals

spawn.c

simple example of posix spawn run date --utc to print current UTC

```
#include <stdio.h>
#include <unistd.h>
#include <spawn.h>
#include <sys/wait.h>

int main(void) {
    pid_t pid;
    extern char **environ;
    char *date_argv[] = {"/bin/date", "--utc", NULL};
    if (posix_spawn(&pid, "/bin/date", NULL, NULL, date_argv, environ) != 0) {
        perror("spawn");
        return 1;
    }

    int exit_status;
    if (waitpid(pid, &exit_status, 0) == -1) {
        perror("waitpid");
        return 1;
    }

    printf("/bin/date exit status was %d\n", exit_status);
    return 0;
}
```

exec.c

simple example of program replacing itself with exec

```
#include <stdio.h>
#include <unistd.h>

int main(void) {

    char *echo_argv[] = {"/bin/echo", "good-bye", "cruel", "world", NULL};
    execv("/bin/echo", echo_argv);

    // if we get here there has been an error
    perror("");
    return 1;
}
```

fork.c

```
#include <stdio.h>
#include <unistd.h>

int main(void) {

    // fork creates 2 identical copies of program
    // only return value is different

    pid_t pid = fork();

    if (pid == -1) {
        // the fork failed, perror will print why
        perror("fork");
    } else if (pid == 0) {
        printf("I know I am the child because fork() returned %d.\n", pid);
    } else {
        printf("I know I am the parent because fork() returned %d.\n", pid);
    }

    return 0;
}
```

fork_exec.c

simple example of classic fork/exec run date --utc to print current UTC
use posix_spawn instead

```c
#include <stdio.h>
#include <unistd.h>
#include <spawn.h>
#include <sys/wait.h>

int main(void) {
    pid_t pid = fork();

    if (pid == -1) {
        // the fork failed, perror will print why
        perror("fork");
    } else if (pid == 0) {
        // child

        char *date_argv[] = {"/bin/date", "--utc", NULL};

        execv("/bin/date", date_argv);

        // execution will not reach here if exec is successful
        perror("execvpe");
        return 1;
    } else {
        // parent

        int exit_status;
        if (waitpid(pid, &exit_status, 0) == -1) {
            perror("waitpid");
            return 1;
        }
        printf("/bin/date exit status was %d\n", exit_status);
    }

    return 0;
}
```

system.c
simple example of system run date --utc to print current UTC

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {

    // system passes string to a shell for evaluation
    // brittle and highly-vulnerable to security exploits
    // system is suitable for quick debugging and throw-away programs only

    int exit_status = system("/bin/date --utc");
    printf("/bin/date exit status was %d\n", exit_status);
    return 0;
}
```

lsld_spawn.c
spawn ls -ld adding as argument the arguments we have been given

```c
#include <stdio.h>
#include <stdlib.h>
#include <spawn.h>
#include <sys/wait.h>

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

    char *ls_argv[argc + 2];
    ls_argv[0] = "/bin/ls";
    ls_argv[1] = "-ld";
    for (int i = 1; i <= argc; i++) {
        ls_argv[i + 1] = argv[i];
    }

    pid_t pid;
    extern char **environ;
    if (posix_spawn(&pid, "/bin/ls", NULL, NULL, ls_argv, environ) != 0) {
        perror("spawn");
        exit(1);
    }

    int exit_status;
    if (waitpid(pid, &exit_status, 0) == -1) {
        perror("waitpid");
        exit(1);
    }

    // exit with whatever status ls exited with
    return exit_status;
}
```

lsld_system.c

spawn ls -ld adding as argument the arguments we have been given

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[]) {
    char *ls = "/bin/ls -ld";
    int command_length = strlen(ls);
    for (int i = 1; i < argc; i++) {
        command_length += strlen(argv[i]) + 1;
    }

    // create command as string
    char command[command_length + 1];
    strcpy(command, ls);
    for (int i = 1; i <= argc; i++) {
        strcat(command, " ");
        strcat(command, argv[i]);
    }

    int exit_status = system(command);
    return exit_status;
}
```

environ.c

print allenvirnoment variables

```c
#include <stdio.h>

int main(void) {
    extern char **environ;

    for (int i = 0; environ[i] != NULL; i++) {
        printf("%s\n", environ[i]);
    }
}
```

getenv.c

simple example of accessing an environment variable

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    char *value = getenv("STATUS");
    printf("Environment variable 'STATUS' has value '%s'\n", value);
    return 0;
}
```

setenv.c

simple example of setting an environment variable

```c
#include <stdio.h>
#include <stdlib.h>
#include <spawn.h>
#include <sys/wait.h>

int main(void) {
    setenv("STATUS", "great", 1);
    char *getenv_argv[] = {"./getenv", NULL};
    pid_t pid;
    extern char **environ;
    if (posix_spawn(&pid, "./getenv", NULL, NULL, getenv_argv, environ) != 0) {
        perror("spawn");
        exit(1);
    }
    return 0;
}
```

spawn_environment.c

simple example of using environment variableto change program behaviour run date -to print time
Perth time printed, due to TZ environment variable

```c
#include <stdio.h>
#include <unistd.h>
#include <spawn.h>
#include <sys/wait.h>

int main(void) {
    pid_t pid;

    char *date_argv[] = {"/bin/date", NULL};
    char *date_environment[] = {"TZ=Australia/Perth", NULL};
    if (posix_spawn(&pid, "/bin/date", NULL, NULL, date_argv, date_environment) != 0) {
        perror("spawn");
        return 1;
    }

    int exit_status;
    if (waitpid(pid, &exit_status, 0) == -1) {
        perror("waitpid");
        return 1;
    }

    printf("/bin/date exit status was %d\n", exit_status);
    return 0;
}
```

busy_wait_for_signal.c

simple example of catching a signal don't compile with dcc

```c
#include <stdio.h>
#include <unistd.h>
#include <signal.h>

void signal_handler(int signum) {
    printf("signal number %d received\n", signum);
}

int main(void) {
    struct sigaction action = {.sa_handler = signal_handler};
    sigaction(SIGUSR1, &action, NULL);

    printf("I am process %d waiting for signal %d\n", getpid(), SIGUSR1);

    // loop waiting for signal
    // bad consumes CPU/electricity/battery
    // sleep would be better

    while (1) {
    }
}
```

wait_for_signal.c

simple example of catching a signal don't compile with dcc

```c
#include <stdio.h>
#include <unistd.h>
#include <signal.h>

void signal_handler(int signum) {
    printf("signal number %d received\n", signum);
}

int main(void) {
    struct sigaction action = {.sa_handler = signal_handler};
    sigaction(SIGUSR1, &action, NULL);

    printf("I am process %d waiting for signal %d\n", getpid(), SIGUSR1);

    // suspend execution for 1 hour
    sleep(3600);
}
```

send_signal.c

simple example of sending a signal

```c
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>


int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <signal> <pid>\n", argv[0]);
        return 1;
    }
    int signal = atoi(argv[1]);
    int pid = atoi(argv[2]);
    kill(pid, signal);
}
```

ignore_control_c.c

simple example of catching a signal don't compile with dcc

```c
#include <stdio.h>
#include <unistd.h>
#include <signal.h>


int main(void) {
    // catch SIGINT which is sent if user types cntrl-d

    struct sigaction action = {.sa_handler = SIG_IGN};
    sigaction(SIGINT, &action, NULL);

    while (1) {
        printf("Can't interrupt me, I'm ignoring ctrl-C\n");
        sleep(1);
    }
}
```

laugh_at_control_c.c

simple example of catching a signal don't compile with dcc

```c
#include <stdio.h>
#include <unistd.h>
#include <signal.h>


void ha_ha(int signum) {
    printf("Ha Ha!\n");
}

int main(void) {
    // catch SIGINT which is sent if user types cntrl-d

    struct sigaction action = {.sa_handler = ha_ha};
    sigaction(SIGINT, &action, NULL);

    while (1) {
        printf("Can't interrupt me, I'm ignoring ctrl-C\n");
        sleep(1);
    }
}
```

stop_with_control_c.c

simple example of catching a signal don't compile with dcc

```c
#include <stdio.h>
#include <unistd.h>
#include <signal.h>

int signal_received = 0;

void stop(int signum) {
    signal_received = 1;
}

int main(void) {
    // catch SIGINT which is sent if user types cntrl-C
    struct sigaction action = {.sa_handler = stop};
    sigaction(SIGUSR1, &action, NULL);

    while (!signal_received) {
        printf("Type ctrl-c to stop me\n");
        sleep(1);
    }
    printf("Good bye\n");
}
```

catch_error.c

simple example of catching a signal don't compile with dcc

```c
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>

void report_signal(int signum) {
    printf("Signal %d received\n", signum);
    printf("Please send help\n");
    exit(0);
}


int main(int argc, char *argv[]) {
    struct sigaction action = {.sa_handler = report_signal};
    sigaction(SIGFPE, &action, NULL);

    // this will produce a divide by zero
    // if there are no command-line arguments
    // which will cause program to receive SIGFPE

    printf("%d\n", 42/(argc - 1));

    printf("Good bye\n");
}
```

spawn_read_pipe.c

simple example using a pipe with posix spawn to capture output from spawned process

```c
#include <stdio.h>
#include <unistd.h>
#include <spawn.h>
#include <sys/wait.h>

int main(void) {
    // create a pipe
    int pipe_file_descriptors[2];
    if (pipe(pipe_file_descriptors) == -1) {
        perror("pipe");
        return 1;
    }

    // create a list of file actions to be carried out on spawned process
    posix_spawn_file_actions_t actions;
    if (posix_spawn_file_actions_init(&actions) != 0) {
        perror("posix_spawn_file_actions_init");
        return 1;
    }

    // tell spawned process to close unused read end of pipe
    // without this - spawned process would not receive EOF
    // when read end of the pipe is closed below,
    if (posix_spawn_file_actions_addclose(&actions, pipe_file_descriptors[0]) != 0) {
        perror("posix_spawn_file_actions_init");
        return 1;
    }

    // tell spawned process to replace file descriptor 1 (stdout)
    // with write end of the pipe
    if (posix_spawn_file_actions_adddup2(&actions, pipe_file_descriptors[1], 1) != 0) {
        perror("posix_spawn_file_actions_adddup2");
        return 1;
    }

    pid_t pid;
    extern char **environ;
    char *date_argv[] = {"/bin/date", "--utc", NULL};
    if (posix_spawn(&pid, "/bin/date", &actions, NULL, date_argv, environ) != 0) {
        perror("spawn");
        return 1;
    }

    // close unused write end of pipe
    // in some case processes will deadlock without this
    // not in this case, but still good practice
    close(pipe_file_descriptors[1]);

    // creae a stdio stream from read end of pipe
    FILE *f = fdopen(pipe_file_descriptors[0], "r");
    if (f == NULL) {
        perror("fdopen");
        return 1;
    }

    // read a line from read-end of pipe
    char line[256];
    if (fgets(line, sizeof line, f) == NULL) {
        fprintf(stderr, "no output from date\n");
        return 1;
    }

    printf("output captured from /bin/date was: '%s'\n", line);

    // close read-end of the pipe
    // spawned process will now receive EOF if attempts to read input
    fclose(f);

    int exit_status;
    if (waitpid(pid, &exit_status, 0) == -1) {
        perror("waitpid");
        return 1;
    }
    printf("/bin/date exit status was %d\n", exit_status);

    return 0;
}
```

read_popen.c

simple example of use to popen to capture output don't compile with dcc - it currently has a bug with popen

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {

    // popen passes string to a shell for evaluation
    // brittle and highly-vulnerable to security exploits
    // popen is suitable for quick debugging and throw-away programs only

    FILE *p = popen("/bin/date --utc", "r");
    if (p == NULL) {
        perror("");
        return 1;
    }

    char line[256];
    if (fgets(line, sizeof line, p) == NULL) {
        fprintf(stderr, "no output from date\n");
        return 1;
    }

    printf("output captured from /bin/date was: '%s'\n", line);

    pclose(p);
    return 0;
}
```

spawn_write_pipe.c

simple example of using a pipe to with posix_spawn to sending input to spawned process

```c
#include <stdio.h>
#include <unistd.h>
#include <spawn.h>
#include <sys/wait.h>

int main(void) {
    // create a pipe
    int pipe_file_descriptors[2];
    if (pipe(pipe_file_descriptors) == -1) {
        perror("pipe");
        return 1;
    }

    // create a list of file actions to be carried out on spawned process
    posix_spawn_file_actions_t actions;
    if (posix_spawn_file_actions_init(&actions) != 0) {
        perror("posix_spawn_file_actions_init");
        return 1;
    }

    // tell spawned process to close unused write end of pipe
    // without this - spawned process will not receive EOF
    // when write end of the pipe is closed below,
    // because spawned process also has the write-end open
    // deadlock will result
    if (posix_spawn_file_actions_addclose(&actions, pipe_file_descriptors[1]) != 0) {
        perror("posix_spawn_file_actions_init");
        return 1;
    }

    // tell spawned process to replace file descriptor 0 (stdin)
    // with read end of the pipe
    if (posix_spawn_file_actions_adddup2(&actions, pipe_file_descriptors[0], 0) != 0) {
        perror("posix_spawn_file_actions_adddup2");
        return 1;
    }


    // create a process running /usr/bin/sort
    // sort reads lines from stdin and prints them in sorted order
    char *sort_argv[] = {"sort", NULL};
    pid_t pid;
    extern char **environ;
    if (posix_spawn(&pid, "/usr/bin/sort", &actions, NULL, sort_argv, environ) != 0) {
        perror("spawn");
        return 1;
    }

    // close unused read end of pipe
    close(pipe_file_descriptors[0]);

    // create a stdio stream from write-end of pipe
    FILE *f = fdopen(pipe_file_descriptors[1], "w");
    if (f == NULL) {
        perror("fdopen");
        return 1;
    }

    // send some input to the /usr/bin/sort process
    //sort with will print the lines to stdout in sorted order
    fprintf(f, "sort\nwords\nplease\nthese\n");

    // close write-end of the pipe
    // without this sort will hang waiting for more input
    fclose(f);

    int exit_status;
    if (waitpid(pid, &exit_status, 0) == -1) {
        perror("waitpid");
        return 1;
    }
    printf("/usr/bin/sort exit status was %d\n", exit_status);

    return 0;
}
```

write_popen.c

simple example of use to popen to capture output don't compile with dcc - it currently has a bug with popen

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {

    // popen passes string to a shell for evaluation
    // brittle and highly-vulnerable to security exploits
    // popen is suitable for quick debugging and throw-away programs only
    //
    // tr a-z A-Z - passes stdin to stdout converting lower case to upper case

    FILE *p = popen("tr a-z A-Z", "w");
    if (p == NULL) {
        perror("");
        return 1;
    }

    fprintf(p, "plz date me\n");

    pclose(p);
    return 0;
}
```