

Module 2

Spawning New Tasks

Overview

- Objectives
- Relevance

Defining a Program

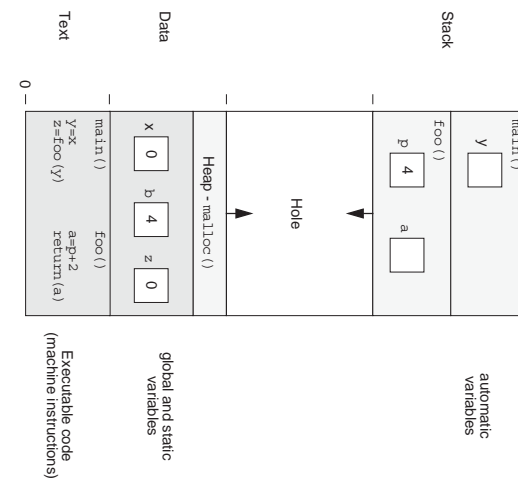
pgm.c

```

1  int x;
2  static int b = 4;
3
4  main() {
5      int y;
6      static int z;
7
8      y = b;
9      z = foo(y);
10 }
11
12 foo( int p ) {
13     int a;
14     a = p + 2;
15     return(a);
16 }
```



Processes as Compared to Procedure Calls





Creating a Process

mysystem.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  main()  {
5
6      int rv;
7
8      /* Sometimes useful for troubleshooting, */
9      /* E.g., do "ps" command inside program and save output */
10     rv = system("ps -le | grep mysystem > /tmp/junk");
11
12     if ( rv != 0 ) {
13         fprintf(stderr, "Something went wrong!\n");
14     }
15 }
```



myfork.c

```
1  #include <sys/types.h>
2  #include <unistd.h>
3  #include <stdio.h>
4
5  main() {
6
7      pid_t pid;
8
9      pid = fork();
10
11     switch(pid) {
12
13         /* fork failed! */
14         case -1:
15             perror("fork");
16             exit(1);
17
18         /* in new child process */
19         case 0:
20             printf("In Child, my pid is: %d\n", getpid());
```



```
21     do_child_stuff();
22     exit(0);
23
24     /* in parent, pid is PID of child */
25     default:
26         break;
27 }
28
29 /* Rest of parent program code */
30 printf("In parent, my child is %d\n", pid);
31 }
32
33 int do_child_stuff() {
34     printf("\t Child activity here \n");
35 }
```



Running a New Program

	Give Absolute Path	Use PATH Variable	New Environment
argv as list	execl()	execlp()	execl_e()
argv as vector	execv()	execvp()	execve()



Running a New Program

```
1  #include <stdio.h>
2  #include <sys/types.h>
3  #include <unistd.h>
4  #include <sys/wait.h>
5
6  main() {
7
8      pid_t  pid;
9
10     pid = fork();
11
12     switch(pid) {
13
14         /* fork failed! */
15         case -1:
16             perror("fork");
17             exit(1);
18         /* in new child process */
19         case 0:
```



```
20     execlp("ls", "ls", "-F", (char *)0);
21     /* why no test? */
22     perror("execlp");
23     exit(1);
24
25     /* in parent, pid is PID of child */
26     default:
27         break;
28 }
29
30 /* Rest of parent program code */
31 wait(NULL);
32 printf("In parent, my child is %d\n", pid);
33 }
```



Terminating a Process

- `exit()`— Flushes buffers and calls `_exit()` to terminate process
- `_exit()`— Closes files and terminates process
- `atexit()`— Stores function for future execution before terminating a process
- `abort()`— Closes files, terminates the process, and produces a core dump for debugging



Terminating a Process

```
1  #include <stdlib.h>
2  #include <unistd.h>
3
4  void cleanup() {
5
6      char *message = "cleanup invoked\n";
7
8      write(STDOUT_FILENO, message, strlen(message));
9  }
10
11 main() {
12
13     /* Register cleanup() as atexit function */
14     atexit(cleanup);
15
16     /* Other things in program done here */
17
18     exit(0);
19 }
```



Cleaning Up Terminated Processes

- `wait()` – Blocks the process until one of its children is ready to have its status reaped.
- `waitpid()` – Allows you to specify which process to wait for and whether to block.



Cleaning Up Terminated Processes

```

1  #include <sys/types.h>
2  #include <sys/wait.h>
3  #include <stdio.h>
4  #include <unistd.h>
5  #include <stdlib.h>
6
7  main() {
8
9      pid_t  pid;
10     int    status;
11
12     /* fork() a child */
13     switch(pid = fork()) {
14
15     case -1:
16         perror("fork");
17         exit(1);
18
19     /* in child */

```



```

20     case 0:
21         execlp("ls", "ls", "-F", (char *)NULL);
22         perror("execlp");
23         exit(1);
24
25     /* parent */
26     default:
27         break;
28     }
29
30     if (waitpid(pid, &status, 0) == -1) {
31         perror("waitpid");
32         exit(1);
33     }
34
35     /* See wstat(5) for macros used with status
36     from wait(2) */
37     if (WIFSIGNALED(status)) {
38         printf("ls terminated by signal %d.\n",
39             WTERMSIG(status));
40     } else if (WIFEXITED(status)) {
41         printf("ls exited with status %d.\n",

```



```

42         WEXITSTATUS(status));
43     } else if (WIFSTOPPED(status)) {
44         printf("ls stopped by signal %d.\n",
45             WSTOPSIG(status));
46     }
47     return 0;
48 }

```



Exercise: Spawning New Tasks

- Objectives
- Tasks
- Discussion
- Solutions