```
/*
 * This is a free program sample that may be reproduced in any form.
 * The author's information should be retained to preserve its identity.
 *
 * Date written: February 15, 1998
 * Written by: Peraphon Sophatsathit
 * Department of Mathematics, Faculty of Science, Chulalongkorn University.
 * email: Peraphon.S@chula.ac.th, http://pioneer.netserv.chula.ac.th/~sperapho
 * Operating Systems (2301371) classnote.
 * Description: This sample program illustrates multitasking process creation
 *              by means of 'fork' and 'exec' system calls.
 */

#include     <stdio.h>
#include     <string.h>
#define      Path         "/usr/bin/date"
#define      Name         "date"

/*
 * function prototype
 */
int    driver(char *, char *);

/*
 * main invocation module
 */
int
main(int ac, char **av)
{
       int    return_code;
       char   Dpath[BUFSIZ], Pname[BUFSIZ];

       switch (ac)
       {
              case 2:
                     strcpy(Dpath, Path);
                     strcpy(Pname, av[1]);
                     break;
              case 3:
                     strcpy(Dpath, av[1]);
                     strcpy(Pname, av[2]);
                     break;
              default:
                     printf("Usage: %s [ [pathName], progName]\n\n", av[0]);
                     return 1;
       }
       return_code = driver(Dpath, Pname);
       printf("driver exits with code = %d\n", return_code);
       return 0;
}
```

```c
/*
 * The driver function spawns a child process and executes a new program
 * which can be any executable file under the child process environment.
 * In the mean time, the parent process waits for its child process to complete.
 */
int
driver(char *path, char *name)
{
	int	pid = 0;
	int	status;

	pid = fork();
	if (pid == 0)
	{
		execl(path, name, NULL);
		printf("exec failed: child %s could not be spawned\n", name);
		return 2;
	}
	else if (pid > 0)
	{
		printf("parent: spawn succeeded!\n");
	}
	else
	{
		printf("fork failed: parent exiting...\n");
		return 1;
	}
	/*
	 * spawning the new child process has succeeded.
	 * Now the parent process will wait for child to complete
	 * before cleaning up.
	 */
	if (wait(&status) < 0)
	{
		printf("child: exec failed\n");
	}
	else
	{
		printf("parent: wait for child to exit\n");
		if (status != 0)
		{
			printf("error in child process: %d\n", status);
		}
		else
		{
			printf("normal termination\n");
		}
	}
	return 0;
}
```