PRCMP PL12

Pointers and Unix syscalls

December 5, 2023

1 Pointers

1. Write a program that has:
   - variables `a` and `b` of type `int`, and
   - the pointer `p` for `int`.

   The program should perform the following sequence of operations.

   OP1: Set `p` pointing to the variable `a`.

   OP2: Assign the number 20 to the content pointed by `p`.

   OP3: Set `p` pointing to the variable `b`.

   OP4: Ask the user to enter a value, and save it in the position pointed by `p`.

   OP5: Print the addresses and contents of the variables `a`, `b` and `p`.

2. Write a program that asks the user for a signed integer and stores it in a variable. Using a `char` pointer, print in hexadecimal each of the bytes (and their respective addresses) that make up the variable.

   A pointer to `char` can be placed to reference an `int` variable through a cast:

   ```
   char * pointer = (char * ) &variable;
   ```

3. Write the `locate_max()` function that finds the largest element in an array and returns the address where that element is located. The function takes as arguments an array of int and the number of elements in that array, as specified in its prototype: `int * locate_max(int v[], int n_elem);`

   Implement the main function in which an array with capacity for 6 integers is created. The user must fill in the array. Then, print the largest element of the array, using the result of the `locate_max()` function.

4. Implement a program in which an array with capacity for 10 integers is created. This array must be filled with the sequence from 1 to 10.

   Finally, you must print the array values from the last to the first element, using a pointer to int.

5. Implement the `absolute_value()` function that transforms an integer value (given by its address) into its absolute value.

   The function returns one of the following values:
   - 0 if the value was positive,
   - 1 if the value was negative,
   - -1 if an error situation was detected

   The function prototype is: `int absolute_value(int * p_value);`

2 Unix system calls

6. On Unix systems, the `printf()` function uses a system call to print a string to standard output. We can use the interface to the `write` syscall to bypass using the `printf()` function.

   (a) Read the man page for the `write()` function: `man 3 write`

   (b) Write the following code snippet in a program and try it out.

```
char string[] = "I am on my way to stdout!\n";
size_t size_of_string;

size_of_string = strlen(string);

write(STDOUT_FILENO, string, size_of_string);

```

7. Type the shell script `i_am_the_process.sh` which prints the process identifier (PID) of the shell that is interpreting it.

```
#!/bin/bash

echo I am process $$ running from the script.
exit 0

```

   Set the appropriate file mode and run the script a few times. Can you observe differences in the output of the different runs?

8. Write the following C program and save it in file `the_program.c`.

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

int main()
{
    printf("Starting program!\n");
    printf("PID: %d\n", getpid());

    printf("I'm going to turn myself into a script!\n");
    execlp("bash", "bash", "i_am_the_process.sh", (char *) 0);

    printf("Something went wrong! :-/\n");

    return 1;
}

```

   (a) Compile this program and execute it in the same directory as script `i_am_the_process.sh`. What happened?

   (b) Read the `getpid` manual page. What is the return value of this function?

   (c) What is the purpose of the `execlp()` function?

   (d) Move the script file to another directory. What happened?

9. Write the following C program and save it in file `father_child.c`.

```
#include <stdio.h>      // printf()
#include <stdlib.h>     // exit()
#include <unistd.h>     // fork(), execlp(), sleep()
#include <sys/types.h>  // pid_t

int main()
{
```

```
8       pid_t pid;
9       int status;
10
11      printf("[%d] I am the father!\n", getpid());
12
13      printf("[%d] I am going to have a baby!\n", getpid());
14
15      pid = fork();
16
17      if (pid == 0)
18      {
19          /* THIS IS THE CHILD. */
20          printf("[%d] I am the child!\n", getpid());
21          sleep(1);
22
23          printf("[%d] I'm going to turn myself into a script!\n", getpid());
24          execlp("bash", "bash", "i_am_the_process.sh", (char *) 0);
25
26          printf("[%d] Something went wrong! :-/\n", getpid());
27          exit(1);
28      }
29
30      /* THIS IS THE FATHER. */
31      printf("[%d] My baby is born!\n", getpid());
32
33      pid = wait(&status);
34      printf("[%d] My baby %d is dead, with status %d!\n", getpid(), pid,
        WEXITSTATUS(status));
35
36      exit(0);
37  }
38
```

(a) Compile this program and execute it in the same directory as script `i_am_the_process.sh`. What happened?

(b) Read the `fork` manual page. What is its purpose? And what is its return value?

(c) What is the purpose of the `wait()` function?

(d) Move the script file to another directory. What happened?