SISTEMAS OPERATIVOS: AYUDANTÍA 2

Ayudante: Yerko Ortiz

Objetivo: reforzar e interactuar conceptos referentes a syscalls

Implementación

Diseñe e implemente un programa en C en el cual los procesos formen una lista, es decir que el padre tiene un proceso hijo, luego este hijo tiene un proceso hijo y así sucesivamente; si uno de los procesos tiene dos o más hijos entonces habrá fallado en la implementación.
La forma más rápida para verificar(de momento) que su implementación es correcta es imprimir los pid y ppid de los procesos, otra forma sería guardar los pid y ppid de los procesos en un archivo de texto y luego hacer un programa en C que verifique sea una lista de procesos.

Compilador mental

Con lápiz, papel y su imaginación, escriba los datos de salida de los programas de a continuación, describa de manera breve lo que sucede en los programas.

Fork

```c
int main(int argv, char **argc)
{
    pid_t pid = fork();
    int status;
    int value = 1;
    if(pid < 0) {
        perror("FORK ERROR\n");
        exit(1);
    } else if(pid == 0) {
        value += value;
        printf("OWN ID %d | PARENT ID %d | OWN VALUE %d\n", getpid(), getppid(), value);
        exit(0);
    } else {
        printf("OWN ID %d | PARENT ID %d | OWN VALUE %d\n", getpid(), getppid(), value);
        wait(&status);
    }
    return 0;
}
```

Wait

```c
int main(void)
{
    pid_t child_pid = fork();
    if (child_pid > 0){
        sleep(1);
        system("ps -l");
        wait(NULL);
        system("ps -l");
    } else {
        exit(0);
    }
    return 0;
}
```

Exec

```
int main(int arcv, char **argc)
{
    char *myargs[3];
    myargs[0] = strdup("cat");
    myargs[1] = strdup("fork1.c");
    myargs[2] = NULL;
    printf("PROCESO MAIN (pid:%d)\n", (int) getpid());
    int child = fork();
    if (child < 0) {
        printf("ERROR\n");
        exit(1);
    } else if (child == 0) { //
        printf("PROCESO HIJO (PID:%d)\n", (int) getpid());
        execvp(myargs[0], myargs);
        printf("CAT DEL ARCHIVO %s\n", myargs[1]);
    } else {
        wait(NULL);
        printf("PROGRAMA TERMINADO\n");
    }
    return 0;
}
```

"Simplicity is a great virtue but it requires hard work to achieve it and education to appreciate it. And to make matters worse: complexity sells better."

Edsger Dijkstra

Exec

```
int main(void)
{
    execl("./hola", "NOMBRE")
        ;
    return 0;
}
```

```
int main(int argv, char **argc)
{
    printf("%d ", getpid());
    char *const cmd[] = {"ps"};
    execvp("/bin/ps", cmd);
    printf("Hola\n");
    return 0;
}
```

Sleep

```
 int main(int argv, char **argc)
{
    char *const cmd[] = {"ps"};
    int pid = fork();
    if (pid > 0) {
        execvp("/bin/ps", cmd);
        printf("in parent process %d", getpid());
    } else if (pid == 0) {
        sleep(2);
        printf("in child process");
        printf("%d ", getpid());
        execvp("/bin/ps", cmd);
        kill(getpid(), SIGKILL);
    }
    return 0;
}
```

Daemon

```c
int main(int argc, char* argv[])
{
    FILE *fp= NULL;
    pid_t process_id = 0;
    pid_t sid = 0;
    process_id = fork();
    if (process_id < 0)
        exit(EXIT_FAILURE);
    else if (process_id > 0)
        exit(EXIT_SUCCESS);
    umask(0); // file mode
    sid = setsid(); // se convierte en el root de un grupo de procesos.
    if(sid < 0)
        exit(EXIT_FAILURE);
    close(STDIN_FILENO);
    close(STDOUT_FILENO);
    close(STDERR_FILENO);
    fp = fopen ("Log.txt", "w+");
    for(int i = 0; i < 1000; ++i) {
        sleep(1);
        fprintf(fp, "%d %d\n", getpid(), i);
        fflush(fp);
    }
    fprintf(fp, "DAEMON TERMINATED\n");
    fclose(fp);
    exit(EXIT_SUCCESS);
}
```

Kill

```c
int main(int argc, char **argv)
{
    if (argc < 2) {
        printf("usage: ./kill PID");
        return -1;
    }
    kill(atoi(argv[1]), SIGKILL);
    return 0;
}
```

"It's not a bug, it's a language feature"

Van Der Linden, C Programming Secrets

SHELL

En los sistemas de tipo UNIX existen interpretes de comandos, entre ellos está bash, zsh, fish, entre otros.
La gracia de esto es hacer la vida más fácil al computín, tener conocimiento en la shell, le otorgará la oportunidad de hacer sus propios scripts o programas que realicen ciertas tareas. En resumen saber de esto le permitirá hacer tareas con menos esfuerzo, como así mismo usar herramientas como SSH, entre otras cosas.
A continuación se le mencionarán ciertos comandos básicos para que usted explore, conforme avance el curso se mencionarán más y nuevas formas de combinarlos para hacer scripts de todo tipo, de momento su tarea será investigar de ellos y usar su imaginación para usarlos en lo que usted estime conveniente:
ls, cd, mkdir, ps, pwd, pbcopy, mv, echo, cat, grep, touch, kill