FIUBA

75-08 Sistemas Operativos 2004

Código de "Procesos"

Ing. Osvaldo Clúa

```
1.  //                                          arg.cc
2.  #include <iostream>
3.  #include <unistd.h>
4.  #include <sys/types.h>
5.  using namespace std;
6.
7.  int main (int argc, char *argp[])
8.  /*    Lista los argumentos */
9.  {
10.         cout<<endl<<"Proceso nro "<<getpid()<<endl;
11.     for (int i=0; argp[i] != NULL; i++)
12.             cout <<"a "<<i<<")"<< argp[i]<<endl;
13. /* acceso al ambiente como variable externa  */
14.     extern char **environ;
15.     for (int i=0; environ[i] != NULL; i++)
16.         cout <<"e "<<i<<")"<< environ[i]<<endl;
17.     exit (0);
18. }
```

```
//                                          envir.cc
1.  #include <iostream>
2.  #include <string>
3.  using namespace std;
4.
5.  int main (int argc, char * argv[], char * envp[])
6.  /* ver los argumentos del comando y las variables de ambiente */
7.  {
8.          /* ver los argumentos   */
9.          for (int i = 0; i < argc; i++)
10.             cout<< "argv["<<i<<"]="<<argv[i]<<endl;
11.         /* ver las variables del ambiente */
12.         for (int i = 0; envp[i] != NULL; i++)
13.              cout<<"envp["<<i<<"]="<< envp[i]<<endl;
14.         exit(0);
15. }
```

```
//                                          fork01.cc
1.  #include <iostream>
2.  #include <unistd.h>
3.  #include <sys/types.h>
4.  using namespace std;
5.
```

```
6.  int main ()
7.  /* ver los pid's que genera un fork */
8.  {
9.      int pidhijo;
10.     cout<<endl<<"El PADRE antes del fork: pid="<<getpid()<<
11.      " group pid= "<<getpgrp()<<endl;
12.     if ( (pidhijo = fork () ) == -1)
13.      {perror("no se puede hacer el fork");exit(1);}
14.      else if (pidhijo == 0)
15.           {/* es el proceso del hijo */
16.           cout<<endl<< "---> Es el HIJO con pid = "<<getpid()<<
17.                " cuyo padre es pid = "<< getppid()<<endl;
18.           cout<<"     El group pid del hijo es = "<< getpgrp()<<endl;
19.           exit(0);
20.           }
21.         else
22.           {/* es el proceso del padre */
23.           cout <<endl<< "Es el PADRE con pid = "<<getpid()<<
24.              " y su hijo es pid = "<<pidhijo<<endl;
25.           exit(0);
26.           }
27. }
```

```
/*                                                    fork02.cc  */
1.  #include<unistd.h>
2.  #include<iostream>
3.  #include<sys/wait.h>
4.  using namespace std;
5.
6.  int main ()
7.  /* ver los pid's que genera un fork, con un padre que espera */
8.  /* espera con wait por el primer hijo que termine y el uso de las macros */
9.  {
10.       int pidhijo, status;
11.       printf ("\n\n");
12.       printf ("El PADRE antes del fork: pid=%d, group pid=%d\n",
13.               getpid(), getpgrp());
14.       if ( (pidhijo = fork () ) == -1)
15.       {
16.           perror("no se puede hacer el fork");
17.           exit(1);
18.       }
19.       else if (pidhijo == 0)
20.           {   /* es el proceso del hijo */
21.               printf ("\n---> Es el HIJO con pid = %d, cuyo padre es pid = %d\n", getpid(),
getppid());
22.               printf ("     El group pid del hijo es = %d\n", getpgrp());
23.               exit(0);
24.           }
25.           else
26.           {   /* es el proceso del padre */
27.               printf ("\nEs el PADRE con pid = %d, y su hijo es pid = %d\n", getpid(), pidhijo);
28.               if (wait(&status) != pidhijo)
29.                   perror("error en el wait ");
30.               if (WIFEXITED(status))
31.                   printf ("termino bien, estado = %d\n", WEXITSTATUS(status));
32.               else if (WIFSIGNALED(status))
```

```
33.                     printf ("termino mal, nro.senial = %d\n", WTERMSIG(status));
34.                 else if (WIFSTOPPED(status))
35.                     printf ("hijo parado, nro senial = %d\n", WSTOPSIG(status));
36.             exit(0);
37.         }
38. }
```

// hacezombie.cc

```
1.  #include<unistd.h>
2.  #include<iostream>
3.  #include<sys/wait.h>
4.
5.  using namespace std;
6.
7.  int main(){
8.  pid_t hijo;
9.    switch (hijo=fork()) {
10.             case -1: {cout << "Error al lanzar el proceso ";exit(3);}
11.             case 0: { // hijo
12.                     cout << "El hijo pid nro "<<getpid()<<" muere"<<endl;
13.                     exit (0);}
14.             default: { // padre
15.                     cout << "El padre pid nro "<<getpid()<<" duerme un poco"<<endl;
16.                     sleep (5);
17.                     cout << "El padre pid nro hace un ps"<<endl;
18.                     system ("ps -o pid,ppid,args,stat");
19.                     system ("pstree -ph micho");
20.                     cout << endl;
21.                     exit(0);}
22.             }
23. }
```

/* lanza.cc */

```
1.  #include<unistd.h>
2.  #include<iostream>
3.  #include<sys/wait.h>
4.  using namespace std;
5.
6.  char *env_init[] = { "USER=yo",  NULL };
7.
8.  int main(){
9.   /* especificar pathname y ambiente */
10.  if (execle("/home/micho/0eje2k3/tpc1/guia2/arg","arg", "argumento1",
11.       "argumento2", (char *) 0,env_init) < 0)
12.           {perror("Error en execle");exit(1);}
13. }
```

/* sali.cc */

```
1.  #include<unistd.h>
2.  #include <iostream>
3.  using namespace std;
4.
5.  void chau(){
6.    cout <<endl<<"*****Adios, me voy*****"<<getpid()<<endl;
7.  }
8.
9.  int main(){
```

```
10.   cout <<"hola, yo soy "<<getpid()<<endl;
11.   if (atexit (chau)<0)
12.     {perror(" No pude registrar chau");exit(1);}
13.   }
```

```
//                                    thread/thr1.cc
1.  #include <pthread.h>
2.  #include <iostream>
3.  #include <unistd.h>
4.  using namespace std;
5.
6.  // En Linux, este codigo indica que cada thread es en
7.  // realidad un proceso separado (esta implementado con clone)
8.  // compilar con -lpthread.
9.
10. void *func(void *arg){
11.   cout <<"soy la thread pthread_self()->"<<pthread_self();
12.   cout <<" pertenezco al proceso getpid() "<<getpid()<<endl;
13.   sleep (10);
14.   return NULL;
15. }
16. int main(){
17.   pthread_t t;
18.   cout << "Soy el proceso principal getpid()->"<<getpid();
19.   cout <<" y mi thread es  pthread_self()->"
20.     <<pthread_self()<<endl;
21.   pthread_create(&t,NULL,func,NULL);
22.   pthread_join(t,NULL);
23. }
```

```
//                                    thread/thr2.cc
1.  #include <pthread.h>
2.  #include <iostream>
3.  #include <unistd.h>
4.  using namespace std;
5.
6.  // En Linux, a pesar de que cada thread es un proceso separado
7.  // se ve que comparte memoria
8.  // compilar con -lpthread.
9.
10. int variable_compartida;
11.
12. void *func(void *arg){
13.   extern int variable_compartida;
14.   for (int i=0; i<10;i++)
15.       variable_compartida++;
16.   return NULL;
17. }
18. int main(){
19.   pthread_t th1,th2;
20.   extern int variable_compartida;
21.   variable_compartida=0;
22.   pthread_create(&th1,NULL,func,NULL);
23.   pthread_create(&th2,NULL,func,NULL);
24.   for(int i=0;i<10;i++)
25.       variable_compartida++;
26.   pthread_join(th1,NULL);
```

```
27.   pthread_join(th2,NULL);
28.   cout<<"variable_compartida vale "<<variable_compartida<<endl;
29. }
```

// thread/fork2.cc

```
1.  #include <pthread.h>
2.  #include <iostream>
3.  #include <unistd.h>
4.  #include <sys/types.h>
5.  #include <sys/wait.h>
6.  using namespace std;
7.
8.  // En Linux, version con fork del programa thr2
9.  // con globales
10.
11. int variable_compartida;
12.
13. void func(){
14.   extern int variable_compartida;
15.   for (int i=0; i<10;i++)
16.       variable_compartida++;
17. }
18. int main(){
19.   extern int variable_compartida;
20.   pid_t p1,p2;
21.   int status;
22.   variable_compartida=0;
23.   if ((p1=fork())==0) {func();exit(0);};
24.   if ((p2=fork())==0) {func();exit(0);};
25.   for(int i=0;i<10;i++)
26.         variable_compartida++;
27.   waitpid(p1,&status,0);
28.   waitpid(p2,&status,0);;
29.   cout<<"variable_compartida vale "<<variable_compartida<<endl;
30. }
```

// thread/thr3.cc

```
1.  #include <pthread.h>
2.  #include <iostream>
3.  #include <unistd.h>
4.  using namespace std;
5.
6.  // En Linux, a pesar de que cada thread es un proceso separado
7.  // se ve que comparte memoria (con algo de debug)
8.  // compilar con -lpthread.
9.
10. int variable_compartida;
11.
12. void *func(void *arg){
13.   extern int variable_compartida;
14.   for (int i=0; i<10;i++)
15.      variable_compartida++;
16.   cout<<"Proceso "<<getpid()<<", thread "<<pthread_self()
17.    <<", variable_compartida "<<variable_compartida<<endl
18.    <<", &variable_compartida "<<&variable_compartida<<endl;
19.   return NULL;
20. }
```

```
int main(){
  pthread_t th1,th2;
  extern int variable_compartida;
  variable_compartida=0;
  pthread_create(&th1,NULL,func,NULL);
  pthread_create(&th2,NULL,func,NULL);
  for(int i=0;i<10;i++)
     variable_compartida++;
  pthread_join(th1,NULL);
  pthread_join(th2,NULL);
  cout<<"variable_compartida vale "<<variable_compartida<<endl;
  cout<<"Proceso "<<getpid()<<", thread "<<pthread_self()
     <<", variable_compartida "<<variable_compartida<<endl
     <<", &variable_compartida "<<&variable_compartida<<endl;

}
```

// thread/fork3.cc

```
#include <pthread.h>
#include <iostream>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
using namespace std;

// En Linux, version con fork del programa thr3
// con globales y algo de debug
// Las direcciones impresas son las mismas (porque?)

int variable_compartida;

void func(){
  extern int variable_compartida;
  for (int i=0; i<10;i++)
     variable_compartida++;
  cout<<"Proceso "<<getpid()<<", variable_compartida "
       <<variable_compartida<<", &variable_compartida "
       <<&variable_compartida<<endl;
}
int main(){
  extern int variable_compartida;
  pid_t p1,p2;
  int status;
  variable_compartida=0;
  if ((p1=fork())==0) {func();exit(0);};
  if ((p2=fork())==0) {func();exit(0);};
  for(int i=0;i<10;i++)
     variable_compartida++;
  waitpid(p1,&status,0);
  waitpid(p2,&status,0);;
  cout<<"variable_compartida vale "<<variable_compartida<<endl;
  cout<<"Proceso "<<getpid()<<", variable_compartida "
       <<variable_compartida<<", &variable_compartida "
       <<&variable_compartida<<endl;
}
```

// thread/thr4.cc

```
1.  #include <pthread.h>
2.  #include <iostream>
3.  #include <unistd.h>
4.  using namespace std;
5.
6.  // En Linux, forma "civilizada" de compartir variables
7.  // (sin globales pero con un show de * y &)
8.  // compilar con -lpthread.
9.
10.
11. void *func(void *arg){
12.   int * par_p=(int * )arg;
13.   for (int i=0; i<10;i++) (*par_p)++;
14.   cout<<"Thread "<<pthread_self()<<", par_p "<<par_p
15.         <<", (*par_p) "<<(*par_p)<<endl;
16.   return NULL;
17. }
18. int main(){
19.   pthread_t th1,th2;
20.   int variable;
21.   variable=0;
22.   pthread_create(&th1,NULL,func,(void *)&variable);
23.   pthread_create(&th2,NULL,func,(void *)&variable);
24.   for(int i=0;i<10;i++)
25.       variable++;
26.   pthread_join(th1,NULL);
27.   pthread_join(th2,NULL);
28.   cout<<"variable vale "<<variable<<", &variable "<<&variable<<endl;
29. }
```

```
1.  #! /usr/bin/perl
2.  #
3.  # Atencion: perl debe haber sido compilado con threads.
4.  # Busque una distribucion asi preparada para probar este ejemplo
5.  #
6.  use English;
7.  use threads;
8.  sub sub1 {
9.         print "En el thread ",threads->self->tid()," proceso $PID \n";
10.  }
11.
12. $thr1 = threads->create(\&sub1);
13. $thr2 = threads->create(\&sub1);
14. @r1=$thr1->join;
15. @r2=$thr2->join;
16. print "En el thread ",threads->self->tid()," proceso $$ \n";
```