ایجاد و مدیریت فرآیندها در لینوکس - قسمت سوم (نسخه PDF)

ایجاد و مدیریت فرآیندها در لینوکس - قسمت سوم (نسخه چاپی)

۶. مثال

در برنامه زیر ابتدا یک پردازه فرزند ایجاد می شود، سپس تا خاتمه فرزند wait می شود و در نهایت PID فرزند به همراه حالت خاتمه اش (به صورت دهدهی و هگزادسیمال) لیست می شود.

```
/* parent.c */ // parent code
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
main()
{
 int pid, state;
 printf("The parent: before the fork()\n");
 if ((pid=fork()) != 0)
  wait(&state);
 else {
  execl("./child","child", NULL);
  perror("Error exec");
 }
 printf("The parent: after fork()\n");
 state = WEXITSTATUS(state);
 printf("PID child=%d; terminated with the code %d=%x\n",pid, state, state);
}
```

```
/* child.c */ // child code
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
main()
{
 int pid;
 printf("The child: begins the execution \n");
 pid=getpid();
```

```
 pid=getpid();
 printf("The son: %d terminates\n", pid);
 exit(10);
}
```

نمونه اجرای برنامه

---

۷. برنامه شماره ۱

برنامه ای بنویسـید که از سه دسـتور fork متوالی اسـتفاده کرده و هشت پردازه ایجاد کند. با اسـتفاده از دسـتورات مربوط به مشـخصات پردازه ها برای هر پردازه، شماره ID و شماره ID ایجاد کننـده آن را به دست آورید. درختی براساس این IDاها رسم کنید که رابطه Parent-child را نشان دهد.

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
void main()
{
pid_t pid;
pid=fork();
printf("\nReturn value of fork call (0 in child proccess & pid of newly created proccess in parent proccess): %d",pid);
printf("\nProccess PID:%d",getpid());
printf("\nParent proccess PID: %d\n",getppid());

pid=fork();
printf("\nReturn value of fork call (0 in child proccess & pid of newly created proccess in parent proccess): %d",pid);
printf("\nProccess PID:%d",getpid());
printf("\nParent proccess PID: %d\n",getppid());

pid=fork();
printf("\nReturn value of fork call (0 in child proccess & pid of newly created proccess in parent proccess): %d",pid);
printf("\nProccess PID:%d",getpid());
printf("\nParent proccess PID: %d\n",getppid());
}
```

نمونه اجرای برنامه

همانطور که در شـکل زیر نشان داده شده است، پردازه پدر قبل از اتمام اجرای فرزندان به پایان رسیده است. به همین دلیل، فرزندان یتیم شده و سیسـتم عامل شماره پردازه ی پدر آن ها را به ۱ تنظیم می کنـد. برای جلوگیری از این اتفاق می توان از تابع wait استفاده کرد. در ادامه این امر نشان داده شده است.

درخت پردازه ها

برنامه پردازه پدر به شکلی دیگر

```
/* parent.c */ // parent code
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
main()
{
 int pid, state;
 int i;
 for(i=0;i<3;i++)
 {
  pid=fork();
  wait(&state);
  //printf("\nfork call return value: %d",pid);
  printf("\nProccess PID:%d",getpid());
  printf("\tParent proccess PID: %d\n",getppid());
 }
}
```

نمونه اجرای برنامه و درخت پردازه ها

---

۸. برنامه شماره ۲

برنامه ای بنویسـید که از یک دستور fork و همچنین تابع wait اسـتفاده کنـد. برنامه بایـد به گونه ای باشد که فرآیند والد از نحوه اجرای فرآیند فرزند مطلع باشد.

توضـیح: این برنامه مانند مثال آورده شده در قسـمت ۶ می باشـد. تنها با اضافه کردن ماکروی WIFEXITED(*pstatus) از اجرای موفقیت آمیز فرزند اطمینان حاصل می کنیم.

کد برنامه:

```
/* parent.c */ // parent code
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
main()
```

```
{
 int pid, state;
 printf("The parent: before the fork()\n");
 if ((pid=fork()) != 0)
  wait(&state);
 else
{
  execl("./child","child", NULL);
  perror("Error exec");
 }
 printf("The parent: after fork()\n");
 if(WIFEXITED(state)==1)
 {
  state = WEXITSTATUS(state);
  printf("PID child=%d; terminated successfuly with the code %d(in decimal)=%x(in hex)\n",pid, state, state);
}
}
```

```
/* child.c */ // child code
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
main()
{
 int pid;
 printf("The child: begins the execution \n");
 pid=getpid();
 printf("The son: %d terminates\n", pid);
}
```

نمونه اجرای برنامه:

نویسنده : **رامین غلامی تقی زاده**

منبع : **انجمن تخصصی فناوری اطلاعات ایران**

اسماعیل بشارتی فرد

وقتی پروسـس های فرزند یتیم می شن، طبق توافق انجام شده در هنگام طراحی لینوکس، پروسـسinit پدر پروسس های یتیم میشه، یعنی شماره پدر پروسس های یتیم برابر ۱ اعلام میشه.

اما من کـد بالا رو توی اوبنتو ۱۶.۰۴LTS کامپایل کردم، به جای اینکه شـماره پدر پروسـس یتیم رو ۱ اعلام کنه، ۱۱۲۰ اعلام می کنه که شماره پروسس upstart هست.

اونطوری که من جستجو کردم، ظاهرا Upstart جایگزین init هست.

می خواستم بپرسم که آیا این امر توجیه پذیر هست یا اینکه کار من دچار اشکال شده؟

مطلب اصلی