Helburuak:
man komandoa ezagutu (eta erabili)
Bash komando interpretea erabiliz prozesuak kudeatu
Bash komando interpretea erabiliz prozesuei seinaleak bidali
Prosezuen sistema-deiak erabiliz prozesuak kudeatu
Prosezuen sistema-deiak erabiliz prozesuei seinaleak bidali

Bash komandoak prozesuak kudeatzeko:

1.- ps komandoak hainbat aukera ditu. Aztertu itzazu -a -u -x aukerak (zer egiten dute, zertarako erabili daitezke?)

-a: Prozesu guztiak aukeratu terminalarekin ez erlazionatutakoak eta sesio liderrak izan ezik.
-u: Aukeratu EUID-aren arabera
-x: Zure prozesu guztiak aukeratu (ps comandoa egin duen erabiltzaileak).

2.-Egin c lengoaian programa bat begizta infinito batekin. Egikaritu duzunez, komando interpretea blokeatuta geratu da. Ireki ezazu beste terminal bat eta prozesua amaitu (kill komandoa erabili)

```
#include <stdio.h>

int main(){
   int bueltaKop = 0;

   while (bueltaKop>=0) {

      bueltaKop++;
      printf("%d",bueltaKop);

   }
}
```

Terminalean:

```
ps -u

kill 4436
```

3.-Lortu ezazu euiti erabiltzaileak egikaritzen ari den prozesuen zerrenda.

```
ps -x
```

4.-Gauza bera egin baina lortu duzun zerrenda bidali ezazu, aldi berean, irteera.txt fitxategira eta sarrera estandarrera (tee komandoaz baliatu zaitezke)

```
ps -x | tee irteera.txt
```

5.-Aurreko programa infinitoa egikaritu. Aurrekoa baino lehentasun txikiagoa esleitu iezaiozu prozesuari. Eta gero, SIGKILL seinalea bidaliozu.

```
Renice +19 4436
kill -SIGKILL 4436
```

6.-”backup.sh” skripta egin /home/euiti/proba kataloagoaren backup-a egiten duena /tmp/homeeuiti.tar.gz izenarekin.
Rsync -arv ~/proba ~/homeeuiti
tar -cvf homeeuiti
gzip homeeuiti hoeeuiti.tar
a)bost minutu barru egikaritzeko, at komandoa erabiliz

at 10:50am March 25
at> ./backup.sh

b)egunero, arratsaldeko 16:00etan.

0 16 * * * ~/backup.sh

c)hileko lehenengo bost egunetan, goizeko 9:00etan.

0 9 * 1,2,3,4,5 * ~/backup.sh

d)orduro, abuztuko ostiraletan.

0 * 5 * 8 ~/backup.sh
e)hilean behin exekuta dadin (man anacron).

Linux-eko APIak prozesuak kudeatzeko:

1. Egikaritu ezazu hurrengo programa:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main(int argc,char **argv){
 int rc=0,status;
 if (argc!=1) {
    printf("uso: %s\n",argv[0]);
    rc=1;
 }
 else if (fork()) {
     wait(&status);
     printf("GURASOA: pid=%8d ppid=%8d user-id=%8d  \n",getpid(), getppid(), getuid());
     printf("Semeak itzuli duen egoera-kodea: %d\n", status);
  }
  else {
      printf("SEMEA: pid=%8d \n", getpid());
          }
  exit(rc);
}
```

eta erantzun hurrengo galdereei:
a) Zer bistaratzen da pantailan?
SEMEA: pid= 4557
GURASOA: pid= 4556 ppid= 4436 user-id= 1000
Semeak itzuli duen egoera-kodea: 0
b) Zer egikaritzen da lehenago, gurasoa edo semea? Zergatik? Umea, gurasoak wait duelako. Semea bukatu arte itxaron egingo du.
c) Hurrengo aginduan, wait(&status), zer da status aldagaian jasotzen dena?

status-ean gurasoak umearen egoera jasoko du eta egoera honen balioa umeak exit egitean itzulitako balioa izango da.

d)Zer itzuliko du exit aginduak errore bat jasoz gero? eta errorerik gabe amaitzen bada programa?
Errorerik badago, 1 itzuliko du exit-ek, bestela 0.

2. Azaldu zer egiten duen hurrengo programak:

// **trapper.c**

```
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
#include <stdio.h>
void trapper(int);
int main(int argc, char *argv[])
{
    int i;
    for(i=1;i<=64;i++){
            signal(i, trapper);}
    printf("Identificador del proceso: %d\n", getpid() );
    pause();
    printf("Continuando...\n");
    return 0;
}
void trapper(int sig)
{
    signal(sig, trapper);
    printf("Señal que he recogido: %d\n", sig);
}
```

2.1 Konpilatu eta egikaritu. Beren PID-a jaso.

2.2 Beste terminal batetik SIGUSR1 seinalea bidali. Zein da erabili duzun komandoa?

kill -SIGUSR1 4747

2.3 Zer gertatzen da prozesu batek tratatu gabeko seinale heltzen zaionean?

2.4 Aldatu aurreko trapper.c 1tik 9arteko seinaleak tratatzeko. Egikaritu ostean SIGUSR1 seinalea bidatzeko. Zein da erabili duzun komandoa?

Kodean 64 zenbakia 9-gatik ordezkatzea da ideia: for(i=1;i<=9;i++) egitea. Modu honetan lehen 9 seinaleak tratatzen dira (ikus kill -l) eta beste seinaleren bat jasoko balitz defektuzko funtzioa exekutatuko luke. Beraz, prozesu batek, besterik ez bada adierazten, defektuz, kill -l -eko seinale guztientzat defektuzko funtzio bat du, askotan programa etetea izango dena.

2.5 Zer gertatu da SIGUSR1 bidali ostean? Zergatik?

Señal definida por el usuario 1

3. Moldatu programa SIGUSR1 seinalea jasoz gero “Kaixo” mezua ateratzeko.

```
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
#include <stdio.h>
void trapper(int);
int main(int argc, char *argv[])
{
    int i;
    for(i=1;i<=64;i++){
            signal(i, trapper);}
    printf("Identificador del proceso: %d\n", getpid() );
    pause();
    printf("Continuando...\n");
    return 0;
}
void trapper(int sig)
{
```

```
    if(sig==10){
            printf(“Kaixo”);
    }
    signal(sig, trapper);
    printf("Señal que he recogido: %d\n", sig);
}
```

3.1. Moldatu mezu bera ateratzeko hurrengo seinaleekin SIGUSR2, SIGCONT, SIGSTOP eta SIGKILL. Posible da hori egitea? Idatzi seinale bakoitzaren portaera.

SIGUSR2: Erabiltzaileak deitutako seinalea
SIGCONT: Jarraitu geldituta bazegoen
SIGSTOP: Prozesua gelditu
SIGKILL: Prozesua hil

Ezin da programa moldatu SIGSTOP eta SIGKILL-ekin lan egiteko, baina SIGUSR2 eta SIGCONT-ekin bai:

```
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
#include <stdio.h>
void trapper(int);
int main(int argc, char *argv[])
{
    int i;
    for(i=1;i<=64;i++){
            signal(i, trapper);}
    printf("Identificador del proceso: %d\n", getpid() );
    pause();
    printf("Continuando...\n");
    return 0;
}
void trapper(int sig)
{
    if(sig==10||sig==12||sig==19||sig==18||sig==9){
            printf(“Kaixo”);
    }
    signal(sig, trapper);
    printf("Señal que he recogido: %d\n", sig);
}
```

4. **Trapper() funtzioaren barruan beharrezkoa da signal funtzioari berriz deitzea?** Ez
5. **Hurrengo killer.c programa konpilatu**

```
// killer.c
#include <sys/types.h>
#include <signal.h>
#include <unistd.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
    pid_t pid;
    int sig;
    if(argc==3)
    {
            pid=(pid_t)atoi(argv[1]);
            sig=atoi(argv[2]);
            kill(pid, sig);
    } else {
            printf("Uso correcto:\n %s pid signal\n",  argv[0]);
```

```
            return -1;
        }
        return 0;
    }
```

5.1. Zer egiten du atoi funtzioak?

Kate bateko punteroa int batean bihurtzen du

5.2. Linuxen, pid_t mota, ze mota da?

Prozesu baten identifikatzailea da (data type)

5.3. Trapper egikaritu. Bere PID jaso. Eta "./killer PID 9" egikaritu ostean. Ze gertatu da eta zergatik?

6. Hurrengo programa alarm.c egikaritu.

```
#include <signal.h>
    #include <unistd.h>
    #include <stdio.h>
    void trapper(int);
    int main(int argc, char *argv[])
    {
            int i;
            signal(14, trapper);
            printf("Identificador proceso: %d\n", getpid() );
            alarm(5);
            pause();
            alarm(3);
            pause();
            for(;;)
            {
                    alarm(1);
                    pause();
            }
            return 0;
    }

    void trapper(int sig)
    {
            signal(sig, trapper);
            printf("RIIIIIIIIING!\n");
}
```

```
gcc -o alarm alarm.c
./alarm
ander@anderSanju:~/Escritorio$ ./alarm
Identificador proceso: 5817
RIIIIIIIIING!
RIIIIIIIIING!
RIIIIIIIIING!
...
```

6.1. Zer egiten dute hurrengo aginduak?

```
signal(14, trapper);
for(;;) {
alarm(1);
pause(); }
```

Segunduro SIGALARM seinalea bidali, bukle infinitu batean.

6.2. Zer egiten du programak?

7. Hurrengo signalfork.c programa egikaritu:

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
```

```
void trapper(int sig)
{
        signal(sig, trapper(sig));
        printf("SIGUSR1\n");
}

int main(int argc, char *argv[])
{
        pid_t padre, hijo;
        padre = getpid();
        signal( SIGUSR1, trapper );
        if ( (hijo=fork()) == 0 )
        { /* hijo */
                sleep(1);
                kill(padre, SIGUSR1);
                sleep(1);
                kill(padre, SIGUSR1);
                sleep(1);
                kill(padre, SIGUSR1);
                sleep(1);
                kill(padre, SIGKILL);
                exit(0);
        }
        else
        { /* padre */
                for (;;);
        }

        return 0;
}
```

7.1. **Programa honetan semeak aitari bidalitako seinaleak ditugu. Baina zer gertatuko zan aitak semeari SIGUSR1 seinalea bidaliz gero? Signal funtzioaren portaera fork egiterakoan semean ere eragin bera du (portaera heredatzen du?). Zer gertatuko da? Aldatu programa aitak semeari SIGUSR1 seinalea bidaltzeko, eta portaera heredatzen den jakiteko.**

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <signal.h>

void trapper(int sig)
{
        signal(sig, trapper(sig));
        printf("SIGUSR1\n");
}

int main(int argc, char *argv[])
{
        pid_t padre, hijo;
        padre = getpid();
        signal( SIGUSR1, trapper );
        if ( (hijo=fork()) == 0 )
        { /* hijo */
                sleep(10);
        }
        else
        { /* padre */
                sleep(2);
```

```
                kill(hijo,SIGUSR1);
        }

        return 0;
}
```

7.2. Aldatu semearen hurrengo lerro biak:

/*hijo */
sleep(10);

Aldatu aitaren lehenengo lerroak:

/* padre */
sleep(2);
kill(hijo, SIGUSR2);
for (;;);

konpilatu eta egikaritu. Blokeatzen da? Zergatik?

Blokeatutzen da, SIGUSR2-en defektuzko funtzio exekutatzen delako. Gurasoak semeari seinalia bidaliko dio semea lotan dagoenean, horrela umea aktibatzen. Hori dela eta, umearen 'kill' funtzioak ez dira exekutatuko eta bukle infinitu bat sortuko da.

8. Hurrengo programak "lana" katalogoa sortzen du. Egin gauza bera gainerako exec funtzioak erabiliz.

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(void)
{char *args[]={"/bin/mkdir","lana",NULL};
if (execve("/bin/mkdir",args,NULL)==-1)
{perror("execve");
  exit(EXIT_FAILURE);}
puts("No debería llegar aqui");
exit(EXIT_SUCCESS);}
```

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(void)
{char *args[]={"/bin/mkdir","lana",NULL};
if (execve("/bin/mkdir",args[0],args[1],args[2],NULL)==-1)
{perror("execve");
  exit(EXIT_FAILURE);}
puts("No debería llegar aqui");
exit(EXIT_SUCCESS);}
```

9. Beheko programak emanda, azaldu "./padre 60 hijo" deiak egiten duena:

reloj.c

```
#include <stdlib.h> /*atoi and exit*/

#include <stdio.h> /*printf*/

#include <unistd.h> /*alarm*/

#include <signal.h> /*signal and SIGALRM*/
```

```
void xtimer(){
printf("time expired.\n");
}
int main(int argc, char *argv[]){
unsigned int sec;
sec=atoi(argv[1]);
printf("time is %u\n",sec);
signal(SIGALRM,xtimer);
alarm(sec);
pause();
return 0;
}
```

hijo.c

```
#include <unistd.h> /*sleep*/
int main(){
sleep(6);
return 0;
}
```

padre.c

```
#include <sys/types.h> /*kill y wait*/
#include <sys/wait.h> /*wait*/
#include <signal.h>/*kill*/
#include <stdlib.h>/*exit*/
#include <stdio.h>/*printf*/
#include <unistd.h>/*fork and exec...*/
#include <time.h> /*time*/
int main(int argc, char *argv[]){
int idHijo, idReloj, id, t1, status1, status2;
id = getpid();
printf("Proceso padre: %d\n", id);
if((idReloj = fork()) == 0) { /* hijo Reloj */
execl("reloj", "reloj", argv[1], NULL);
}
if((idHijo = fork()) == 0) {
execv(argv[2], &argv[2]);
}
printf("Proceso hijo Perezoso: %d\n", idHijo);
```

```
printf("Proceso hijo Reloj: %d\n", idReloj);
t1 = time(0);
if((id = wait(&status1)) == idHijo) {
kill(idReloj, SIGKILL);
//Si el hijo ha cambiado de estado, antes de la ejecución del wait
//entonces la llamada a wait retornará inmediatamente con su estado
wait(&status2);
} else {
kill(idHijo, SIGKILL);
wait(&status2);
status1 = 1;}
t1=time(0)-t1;
printf("Tiempo del proceso hijo : %d\n", t1);
exit(status1);
}
```

Prozesua abiaraziko da gurasoaren bidez. Programaren exekuzio denbora minimoa (lehen argumentua) pasatzen baldin bada programa bukatu baino lehen, gurasoa semea amaituko du eta ondoren bere id eta prorgamaren pantailan idatziko ditu. Amaitzerakoan gurasoa bere iraupena idatziko du eta itzulera kodea.