

1. test01 test02

2. redirecteaza iesirea comenzii “command” in /dev/fd/60 a carei iesire de eroare o redirecteaza catre d/dev/fd/61

3. rezultatul va fi: def.o cauta.o calc.o

Algoritmul utilizat pentru substitutie este urmãtorul este urmãtorul: se evalueazã expresia ${SRCS}

apoi se cautã sirul de caractere ce urmeazã în definitie dupã caracterul ‘:’ si este înlocuit cu sirul de

caractere de dupã semnul ‘=’.

4. Pentru ca procesul părinte să aştepte terminarea execuţiei unui proces fiu, procesul părintetrebuie să execute apelul sistem wait().Apelul sistem wait returnează PID-ul procesului fiu ce şi-a încheiat execuţia. Parametrul careîl primeşte apelul sistem wait este o referinţă către o locaţie care va primi valoarea stării de terminarea procesului fiu.O altă versiune a apelului sistem wait este waitpid. Prin parametrul options oferă facilităţi suplimentare. De exemplu, în urma unui apel waitpidprocesul părinte nu îşi suspendă execuţia dacă nici un proces fiu nu are disponibilă starea de ieşire.

Mai pe scurt cu wait asteptam ca orice copil sa isi incheie executia iar cu waitpid ca un anume copil sa isi incheie executia.

5.

int main(void)

{

char readbuffer;

pipe(pfd21);

pipe(pfd13);

if(fork()) //P1

{

close(pfd21[1]);

close(pfd13[0]);

while(read(pfd21[0],&readbuffer,1))

write(pfd13[1],&readbuffer,1);

exit(0);

}

else

{

if(fork()) //P2

{

close(pfd21[0]);

char mesaj=”Hello”;

write(pfd21[1],&mesaj,strlen(mesaj)+1);

exit(0);

}else //P3

{

char mesaj;

close(pfd13[1]);

while(read(pfd13[0],&mesaj,1));

}

}

return 0;

}

6.

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

#include <string.h>

void\* print\_message( void \*ptr ){

char \*message;

message = (char \*) ptr;

int end;

end = strlen(message) - 1;

int x;

for (x = end; x >= 0; --x) {

printf("%c", message[x]);

}

return NULL;

}

int main(){

pthread\_t thread1;

char \*message = "Hello World!";

if (pthread\_create( &thread1, NULL, &print\_message,

(void\*)message)){

fprintf(stderr,"thread error\n");

exit(1);

}

sleep(1);

exit(0);

}

Shell:

#!/bin/bash

script() {

if test -d "$1"

then

for i in "$1"/\*

do

counter=0

for j in "$1"/\*

do

if test "$j"="$i"

then counter=`expr $counter + 1`

fi

done

if test $counter -gt 1

then echo $i $counter

fi

done

fi }

script $1

C:#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

#include <dirent.h>

#include <string.h>

#include <fcntl.h>

#include <signal.h>

pid\_t pid\_fiu;

int pfd1[2],pfd2[2];

int numara=0;

void handler(int sig)

{

numara++;

}

void handler2(int sig)

{

printf("%d",numara);

}

void parcurge\_dir(char \*pwd, char \*filename)

{

struct stat buf;

struct dirent \*entry;

DIR \*director;

director=opendir(pwd);

char dirAux[1024], buffer[1024], cale[1024];

while (entry=readdir(director))

{

if ((strcmp(entry->d\_name,".")!=0)&&(strcmp(entry->d\_name,"..")!=0))

{

strcpy(dirAux,pwd);

strcat(dirAux,"/");

strcat(dirAux, entry->d\_name);

stat(dirAux,&buf);

if(S\_ISDIR(buf.st\_mode))

{

parcurge\_dir(dirAux, filename);

}

else{

if((strcmp(entry->d\_name,filename)==0))

{

strcpy(cale,dirAux);

strcat(cale,"/");

strcat(cale, filename);

write(pfd1[1],cale,strlen(cale)+1);

}}

}

}

}

int main(int argc, char \*argv[])

{

if (argc!=3)

return 1;

char readbuffer;

pipe(pfd1);

if((pid\_fiu=fork())==-1)

{

perror("Nu se poate crea procesul fiu");

exit(1);

}

if(pid\_fiu==0)

{//P2

close(pfd1[1]);

while(read(pfd1[0],&readbuffer,1));

ssize\_t readByte;

int fd\_read;

fd\_read=open(readbuffer,O\_RDONLY);

do

{

readByte=read(fd\_read,&buffer,1);

if (strcmp(readByte,argv[2]))

{

kill((int)getppid(),SIGUSR1);

}

}while (readByte==8192);

kill((int)getppid(),SIGUSR2);

exit(0);

}

else

{//P1

char cwd[1024];

signal(SIGUSR1, handler);

signal(SIGUSR2, handler2);

getcwd(cwd,sizeof(cwd));

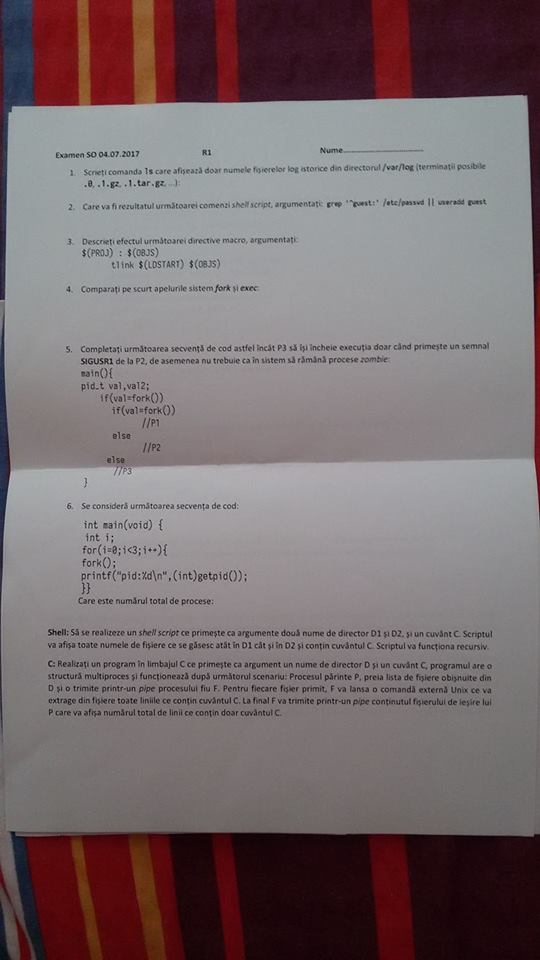
close(pfd1[0]);

parcurge\_dir(cwd,argv[1]);

}

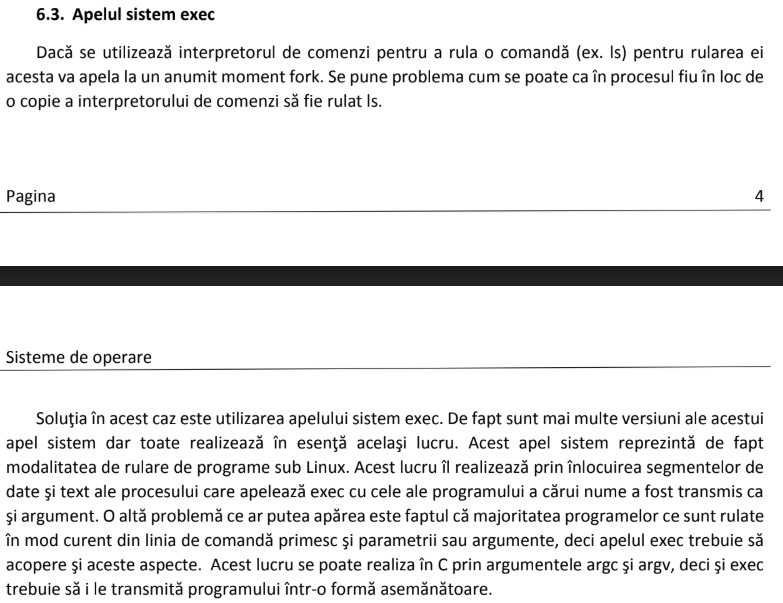
return 0;

}



1. ls /var/log | grep “\.0$\|\.1\.gz$\|\.1\.tar\.gz$”
2. cauta in /etc/passwd toate intrarile care incep cu guest: iar daca userul nu exista il adauga.
3. Aceasta directiva macro creeaza link-ul referit in $(PROJ) din fisierele obiect referite in $(OBJS) folosind linker-ul tlink. Prin $(LDSTART) sunt definite flaguri suplimentare.
4. Fork:Practic acest apel realizeaza un nou proces, numit proces fiu, iar procesul care a apelat fork devineproces parinte a noului proces creat. Cele doua procese sunt identice în termeni de continut alsegmentelor lor de date si text si aproape identice în ceea ce priveste segmentele sistem. Singurelediferente între astfel de procese apar la nivelul unor atribute care trebuie sa fie diferite (cum ar fi PIDcare trebuie sa fie unic pentru fiecare proces în parte). Odata ce procesul fiu a fost creat, ambeleprocese,parinte si fiu, îsi continua executia din interiorul apelului fork. Adica urmatoarea actiunepentru fiecare proces în parte este sa paraseasca fork, fiecare returnând o valoare diferita.

Exec:



1. pid\_t pid1, pid2;

void handler(sigNum)

{

exit(0);

}

int main()

{

if(pid1=fork())

{

if(pid2=fork())

{//P1

exit(0);

}

else

{//P2

wait(10);

kill((int)getppid(),SIGUSR1);

exit(0);

}

}

else

{//P3

signal(SIGUSR1,handler);

while(1);

}

return 0;

}

1. Mie imi creeaza 8…

Shell:

#!/bin/bash

script()  
{

if test -d "$1"  
then

for i in "$1"/\*  
do

if test -f "$i"  
then

contine=`grep "$3" $1`  
if test "$contine" -ge 1  
then

for j in "$2"/\*  
do

if test -f "$j"  
then

if test "$j"="$i"  
then

contine=`grep "$3" $1`  
if test "$contine" -ge 1  
then  
echo "$j"  
fi

fi  
fi

done  
fi

fi  
done

fi  
}

script $1 $2 $3

C:#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

#include <dirent.h>

#include <string.h>

#include <fcntl.h>

#include <signal.h>

pid\_t pid\_fiu;

int pfd1[2],pfd2[2];

int numara=0;

void parcurge\_dir(char \*pwd)

{

struct stat buf;

struct dirent \*entry;

DIR \*director;

director=opendir(pwd);

char dirAux[1024], buffer[1024], cale[1024];

while (entry=readdir(director))

{

if ((strcmp(entry->d\_name,".")!=0)&&(strcmp(entry->d\_name,"..")!=0))

{

strcpy(dirAux,pwd);

strcat(dirAux,"/");

strcat(dirAux, entry->d\_name);

stat(dirAux,&buf);

if(S\_ISREG(buf.st\_mode))

{

strcpy(cale,dirAux);

strcat(cale,"/");

strcat(cale, filename);

write(pfd1[1],cale,strlen(cale)+1);

}

}

}

}

int main(int argc, char \*argv[])

{

if (argc!=3)

return 1;

char readbuffer;

pipe(pfd1);

pipe(pfd2);

if((pid\_fiu=fork())==-1)

{

perror("Nu se poate crea procesul fiu");

exit(1);

}

if(pid\_fiu==0)

{//P2

close(pfd1[1]);

close(pfd2[0]);

while(read(pfd1[0],&readbuffer,1024))

{

system("grep "\<argv[2]\>" readbuffer");

write(pfd1[1],system,strlen(system)+1);

}

exit(0);

}

else

{//P1

close(pfd1[0]);

close(pfd2[1]);

parcurge\_dir(argv[1]);

char fisier[1024];

read(pfd2[0],&fisier,1024);

system("wc -l $fisier");

}

return 0;

}