Esame del 20/01/2023 (GitHub)

Esercizio 1

Si supponga di avere un file "sup_groups.txt" le cui righe contengono informazioni su gruppi unix strutturate nel seguente modo: "nome_gruppo:password:id_gruppo:utente1,utente2..."

Di seguito viene riportato un esempio:

```
adm:x:4:syslog,adm1
admins:x:1006:adm2,adm12,manuel
ssl-cert:x:122:postgres
alan2:x:1009:aceto,salvemini
conda:x:1011:giovannelli,galise,aceto,caputo,haymele,salvemini,scala,adm2,adm12
adm1Group:x:1022:adm2,adm1,adm3
docker:x:998:manuel
```

Utilizzando opportuni comandi in concatenazione si eseguano le seguenti operazioni:

(a) *Elencare i nomi di tutti gli utenti del file*

```
awk -F : '{ print $4 }' sup_groups.txt
```

(b) *Contare il numero di utenti appartenenti al gruppo "admins" e "adm"*

```
awk -F : '{ if($1=="admins" || $1=="adm") print $4 }' sup_groups.txt | awk -F , '{ SOMMA+=NF } END { print SOMMA }'
```

(c) *Elencare il GID dei gruppi con almeno 2 utenti*

```
awk -F , '{ if(NF>2) print }' sup_groups.txt | awk -F : '{ print $3 }'
```

(d) *Elencare il gruppo con il maggior numero di utenti*

```
awk -F , 'BEGIN{ MAX_USER=0; ID_GROUP=0 } { if(NF>MAX_USER) { MAX_USER=NF; ID_GROUP=$1 } print ID_GROUP } END { print ID_GROUP}' sup_groups.txt  | awk -F : 'END { print $3 }'
```

Esercizio 2

Si realizzi uno script di shell BASH "groups", che accetta come argomento un file "groups_file.txt" strutturato nel seguente modo:

```
adm:x:4:syslog,adm1
admins:x:1006:adm2,adm12,manuel
ssl-cert:x:122:postgres
alan2:x:1009:aceto,salvemini
conda:x:1011:giovannelli,galise,aceto,caputo,haymele,salvemini,scala,adm2,adm12
adm1Group:x:1022:adm2,adm1,adm3
docker:x:998:manuel
```

che:

(a) stampa il numero massimo di campi di una linea in un dato file
(b) crea una sottodirectory per ogni gruppo presente nel file, dando accessi di lettura e scrittura agli utenti dei gruppi "adm" e "admins"
(c) crea un file per ogni sotto directory contenente gli utenti che appartengono a quel gruppo riga per riga

```bash
#!/bin/bash

# Verifica che sia stato fornito un file come argomento
if [ "$#" -ne 1 ]; then
    echo "Usage: $0 <groups_file>"
    exit 1
fi

groups_file="$1"
```

```bash
# (a) Stampa il numero massimo di campi di una linea in un dato file
max_fields=$(awk -F':' '{print NF}' "$groups_file" | sort -n | tail -n 1)
echo "(a) Numero massimo di campi in una linea: $max_fields"

# (b) Crea una sottodirectory per ogni gruppo e assegna permessi agli utenti di "adm" e "admins"
while IFS=":" read -r group_name x gid users; do
    dir_name="subdir_$group_name"
    mkdir -p "$dir_name"
    group_members=$(echo "$users" | sed 's/,/ /g')

    # Imposta i permessi sulla directory
    chmod 770 "$dir_name"
    chown :"$group_name" "$dir_name"

    # Imposta i permessi sugli utenti di "adm" e "admins"
    for user in $group_members adm admins; do
        usermod -aG "$group_name" "$user"
        chmod +rw "$dir_name" # Aggiunge permessi di lettura e scrittura
    done
done < "$groups_file"


# (c) Crea un file per ogni sottodirectory contenente gli utenti del gruppo riga per riga
while IFS=":" read -r group_name x gid users; do
        dir_name="subdir_$group_name"
        echo "$users" | tr ',' '\n' > "$dir_name/users.txt"
done < "$groups_file"

echo "Script completato con successo."
```

Esercizio 3

Si implementi un programma C che accetta come argomento il path di un file “f_input”, una stringa “tabu”, una coppia di interi positivi “i” e “j” con i < j.

Il programma stampa a video il contenuto del file di input sostituendo tutte e sole le occorrenze di “tabu”, a partire dalla linea “i-esimo” fino al “j-esimo”, con una stringa contenente tanti asterischi quanti sono i caratteri della stringa “tabu”. Se il file di input contiene meno di “j” linee, la sostituzione avviene fino alla fine del file.

```c
#include "unistd.h"
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "stdbool.h"


int main(int argc, char **argv) {

    FILE *file = fopen(argv[1], "rw+");
    char *tabu = argv[2];
    size_t len = strlen(tabu);
    int i = atoi(argv[3]);
    int j = atoi(argv[4]);

    if (file == NULL || i >= j || i < 0 || j < 0 || argc != 5) printf("Errore\n"), exit(1);

    fseek(file, i, SEEK_SET);

    while(1) {
        if (ftell(file) == j) break;
        char c = fgetc(file);
        if (c == EOF) break;
        if (c == tabu[0]) {
            bool found = true;
            for (int k = 1; k < len; k++) {
                if (fgetc(file) != tabu[k]) {
                    fseek(file, -k, SEEK_CUR);
                    found = false;
```

```
                    break;
                }
            }
            if (found) {
                printf("Trovato %s\n", tabu);
                fseek(file, -len+1, SEEK_CUR);
                for (int k = 0; k < len; k++) {
                    fprintf(file, "*");
                }
            }
        }
    }

    fclose(file);
    return 0;

}
```

Esame del 09/02/2023 (GitHub)

Esercizio 1

A. *Utilizzando awk si scriva un comando che stampi una lista dei file presenti nella directory corrente mostrando solo dimensione e nome.*

```
ls -l | awk '{print $5, $9}'
```

B. *Si calcoli la dimensione occupata in totale dai file regolari con dimensione maggiore di 1024 byte nella directory corrente*

```
ls -l | awk 'BEGIN {sum=0} { if ($5 > 1024 && substr($1,1,1) == "-") sum+=$5} END {print "sum="sum}'
ls -l | awk 'BEGIN {sum=0} { if ($5 > 1024 && $1 ~ /^-/) sum+=$5} END {print "sum="sum}'
```

C. *Si faccia in modo che il comando stampi solo i file maggiori di 1024 byte*

```
ls -l | awk '{ if ($5 > 1024) print $0 }'
```

D. *Trovare i file non acceduti negli ultimi 30 giorni*

```
ls -l --time-style=+%s | awk 'BEGIN { now=systime() } { if(now-$6 < 60*60*24*30) print $0 }'
```

Esercizio 2

Si realizzi uno script di shell BASH "`menu`", che accetta come argomento un file "`listino.txt`" strutturato nel seguente modo:

```
codice   quantità   costo
01953       2        15
07934       1        20
084Gd      10        30
9038H       1         5
```

e che implementi le seguenti funzioni accessibili da un menu:

a) *Cerca - Chiede all'utente una stringa da ricercare all'intero del listino ed effettua la ricerca*
b) *Aggiungi - Chiede all'utente il codice del prodotto da aggiungere (primo campo del listino) e la quantità di articoli desiderati, verifica le scelte effettuate e le memorizza in un file carrello*
c) *Elimina - un prodotto dal carrello*

```
#!/bin/bash
echo "Benvenuto in menù"
if [ $# -ne 1 ]
then
     echo "Utilizzo: $0 <nomefile>"
     exit 1
fi

if [ ! -f $1 ]
then
     echo "Il file $1 non esiste"
     exit 2
fi

echo "Cosa vuoi fare?"
echo "1) Cerca"
echo "2) Aggiungi"
echo "3) Elimina"
echo "4) Esci"
read scelta

case $scelta in
     # Cerca
     1) echo "Inserisci il codice"
         read codice
         grep $codice $1
         ;;
     # Aggiungi
     2) echo "Inserisci il codice"
```

```
        read codice
        if [ $(grep -c $codice $1) -eq 0 ]
        then
            echo "Il codice non esiste"
            exit 3
        fi
        echo "Inserisci la quantità"
        read quantita
        quantita_file=$(grep "$codice" $1 | awk '{print $2}')
        if [ $quantita_file -lt $quantita ]
        then
            echo "Non ci sono abbastanza prodotti"
            exit 4
        fi
        echo $codice $quantita $costo >> carrello.txt
        ;;
    # Elimina
    3) echo "Inserisci il codice da eliminare"
        read codice
        sed -i "/$codice/d" carrello.txt
        ;;
    # Esci
    4) echo "Arrivederci"
        exit 0
        ;;
    # Scelta non valida
    *) echo "Scelta non valida"
        exit 3
        ;;
esac
```

Esercizio 3

Si scriva un programma in C che prende in input i seguenti argomenti: esame i j f C1 C2
Dove esame è il nome dell'eseguibile, i e j sono due interi positivi, f il nome del file, C1 e C2 due caratteri. Il processo padre dovrà creare due processi P1 e P2, il processo P1 dovrà cercare le istanze di C1 nel file f, mentre il processo P2 dovrà cercare le istanze di C2 in f. Quando uno dei due processi trova il carattere, lo segnala al padre. Il padre si mette in attesa che i figli cercano i caratteri, quando riceve il segnale chiamerà una funzione "notifica" che stampa il PID del figlio. Quando uno dei figli trova il carattere, l'applicazione termina.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>

void P_Func(char* f, char C){
    printf("Il processo %d inizia\n", getpid());
    int fd = open(f, O_RDONLY);
    char c;
    do read(fd, &c, sizeof(char));
    while(c!=C && c!=EOF);
}

void notifica(pid_t P_Winner){
    printf("Il processo %d ha vinto\n", P_Winner);
}

void Padre_Func(pid_t P1_PID, pid_t P2_PID){
    pid_t P_Winner = wait(NULL);
    if(P_Winner==P1_PID) kill(P2_PID, SIGKILL);
    else kill(P1_PID, SIGKILL);
    notifica(P_Winner);
}

int main(int argc, char** argv){
    if(argc!=6) printf("Usage: %s <i> <j> <f> <C1> <C2>\n", argv[0]), exit(1);
```

```
    char* f = argv[3];
    char C1 = argv[4][0];
    char C2 = argv[5][0];

    pid_t P1_PID = fork();
    if(P1_PID==0) P_Func(f, C1);
    else{
        pid_t P2_PID = fork();
        if(P2_PID==0) P_Func(f, C2);
        else Padre_Func(P1_PID, P2_PID);
    }

}
```

Esame del 09/02/2023 (GitHub)

Esercizio 1

A. *Utilizzando awk si scriva un comando che stampi una lista dei file presenti nella directory corrente mostrando solo dimensione e nome.*

```
ls -l | awk '{print $5, $9}'
```

B. *Si calcoli la dimensione occupata in totale dai file regolari con dimensione maggiore di 1024 byte nella directory corrente*

```
ls -l | awk 'BEGIN {sum=0} { if ($5 > 1024 && substr($1,1,1) == "-") sum+=$5} END {print "sum="sum}'
ls -l | awk 'BEGIN {sum=0} { if ($5 > 1024 && $1 ~ /^-/) sum+=$5} END {print "sum="sum}'
```

C. *Si faccia in modo che il comando stampi solo i file maggiori di 1024 byte*

```
ls -l | awk '{ if ($5 > 1024) print $0 }'
```

D. *Trovare i file non acceduti negli ultimi 30 giorni*

```
ls -l --time-style=+%s | awk 'BEGIN { now=systime() } { if(now-$6 < 60*60*24*30) print $0 }'
```

Esercizio 2

Si realizzi uno script di shell BASH "`menu`", che accetta come argomento un file "`listino.txt`" strutturato nel seguente modo:

```
codice   quantità   costo
01953       2        15
07934       1        20
084Gd      10        30
9038H       1         5
```

e che implementi le seguenti funzioni accessibili da un menu:

a) *Cerca - Chiede all'utente una stringa da ricercare all'intero del listino ed effettua la ricerca*
b) *Aggiungi - Chiede all'utente il codice del prodotto da aggiungere (primo campo del listino) e la quantità di articoli desiderati, verifica le scelte effettuate e le memorizza in un file carrello*
c) *Elimina - un prodotto dal carrello*

```
#!/bin/bash
echo "Benvenuto in menù"
if [ $# -ne 1 ]
then
     echo "Utilizzo: $0 <nomefile>"
     exit 1
fi

if [ ! -f $1 ]
then
     echo "Il file $1 non esiste"
     exit 2
fi

echo "Cosa vuoi fare?"
echo "1) Cerca"
echo "2) Aggiungi"
echo "3) Elimina"
echo "4) Esci"
read scelta

case $scelta in
     # Cerca
     1) echo "Inserisci il codice"
         read codice
         grep $codice $1
         ;;
     # Aggiungi
     2) echo "Inserisci il codice"
```

```
        read codice
        if [ $(grep -c $codice $1) -eq 0 ]
        then
            echo "Il codice non esiste"
            exit 3
        fi
        echo "Inserisci la quantità"
        read quantita
        quantita_file=$(grep "$codice" $1 | awk '{print $2}')
        if [ $quantita_file -lt $quantita ]
        then
            echo "Non ci sono abbastanza prodotti"
            exit 4
        fi
        echo $codice $quantita $costo >> carrello.txt
        ;;
    # Elimina
    3) echo "Inserisci il codice da eliminare"
        read codice
        sed -i "/$codice/d" carrello.txt
        ;;
    # Esci
    4) echo "Arrivederci"
        exit 0
        ;;
    # Scelta non valida
    *) echo "Scelta non valida"
        exit 3
        ;;
esac
```

Esercizio 3

Si scriva un programma in C che prende in input i seguenti argomenti: esame i j f C1 C2
Dove esame è il nome dell'eseguibile, i e j sono due interi positivi, f il nome del file, C1 e C2 due caratteri. Il processo padre dovrà creare due processi P1 e P2, il processo P1 dovrà cercare le istanze di C1 nel file f, mentre il processo P2 dovrà cercare le istanze di C2 in f. Quando uno dei due processi trova il carattere, lo segnala al padre. Il padre si mette in attesa che i figli cercano i caratteri, quando riceve il segnale chiamerà una funzione "notifica" che stampa il PID del figlio. Quando uno dei figli trova il carattere, l'applicazione termina.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>

void P_Func(char* f, char C){
    printf("Il processo %d inizia\n", getpid());
    int fd = open(f, O_RDONLY);
    char c;
    do read(fd, &c, sizeof(char));
    while(c!=C && c!=EOF);
}

void notifica(pid_t P_Winner){
    printf("Il processo %d ha vinto\n", P_Winner);
}

void Padre_Func(pid_t P1_PID, pid_t P2_PID){
    pid_t P_Winner = wait(NULL);
    if(P_Winner==P1_PID) kill(P2_PID, SIGKILL);
    else kill(P1_PID, SIGKILL);
    notifica(P_Winner);
}

int main(int argc, char** argv){
    if(argc!=6) printf("Usage: %s <i> <j> <f> <C1> <C2>\n", argv[0]), exit(1);
```

```
    char* f = argv[3];
    char C1 = argv[4][0];
    char C2 = argv[5][0];

    pid_t P1_PID = fork();
    if(P1_PID==0) P_Func(f, C1);
    else{
        pid_t P2_PID = fork();
        if(P2_PID==0) P_Func(f, C2);
        else Padre_Func(P1_PID, P2_PID);
    }

}
```

Esame del 06/03/2023 (GitHub)

Esercizio 1

Utilizzando opportuni comandi in concatenazione si eseguano le seguenti operazioni:

(a) *Utilizzando awk si scriva un comando che stampi una lista dei file presenti nella directory corrente mostrando solo nome e proprietario.*

```
ls -l | awk '{print $9" "$3}'
```

(b) *Si calcoli la dimensione occupata in totale dai file regolari con dimensione inferiore di 1024 byte nella directory corrente.*

```
ls -l | awk '{ if($1 ~ /^-/ && $5<1024) somma=somma+$5 } END {print somma}'
```

(c) *Dato un file "parole.txt" stampa solo le linee con più di 10 caratteri.*

```
awk --field-separator= '{if(NF>10) print $0}' parole.txt
```

(d) *Impostando una variabile d'ambiente LIST: perm, link, user, group, date (può anche essere inizializzata al di fuori del comando stesso) visualizzi il listato dei file nella directory corrente con il campo corrispondente*

```
export LIST="perm,link,user,group,date"
ls -l | awk -v list="$LIST" '{split(list, fields, ","); print $fields[1], $fields[2], $fields[3],
$fields[4], $fields[5]}'
```

Esercizio 2

Si realizzi uno script di shell BASH "`menu`", che implementi le seguenti funzioni accessibili da un menu:

(a) Aggiungi verifica - Chiede all'utente gli elementi: giorno, mese, anno, nome studente, voto. E inserirli in un file di nome "`verifica`". Il file deve essere creato sola la prima volta, chiamate successive, dovranno aggiungere nuove righe allo stesso file.
(b) Conta - Chiede all'utente il mese e lo studente, e conta il numero di prove effettuate nel mese per lo studente dato.
(c) Media - Chiede all'utente lo studente, e calcola la media dei voti delle verifiche date dallo studente.

Si rappresentino i mesi con una stringa di tre caratteri (gen, feb, mar, ecc.)

```
#!/bin/bash

scelta=0
while [ $scelta -ne 4 ];
do
    echo "Scegli un'opzione"
    echo "1) Aggiungi verifica"
    echo "2) Conta"
    echo "3) Media"
    echo "4) Uscita"
    read scelta
    case $scelta in
        1) echo "Inserisci il giorno: "
           read giorno
           echo "Inserisci il mese nel formato di 3 lettere (gen feb mar): "
           read mese
           echo "Inserisci l'anno: "
           read anno
           echo "Inserisci il nome dello studente: "
           read nome
           echo "Inserisci il voto: "
           read voto
```

```
            echo $giorno,$mese,$anno,$nome,$voto >> verifica
            ;;
        2) echo "Inserisci il nome dello studente: "
            read nome
            echo "Inserisci il mese in cui contare le prove: "
            read mese
            risultato=$(awk -F, -v mese="$mese" -v nome="$nome" 'BEGIN{somma=0} {if($2==mese && $4==nome)
somma=somma+1} END {print somma}' verifica)
            echo "In $mese ci sono $risultato prove per $nome"
            ;;
        3) echo "Inserisci il nome dello studente: "
            read nome
            somma=$(awk -F, -v nome="$nome" 'BEGIN{somma=0} {if($4==nome) somma=somma+$5} END {print somma}'
verifica)
            numero=$(awk -F, -v nome="$nome" 'BEGIN{somma=0}{if($4==nome) somma=somma+1} END {print somma}'
verifica)
            media=$((somma/numero))
            echo "La media è: $media"
            ;;
        4) echo "Uscita"
            ;;
        *) echo "Scelta non valida"
            ;;
    esac
done
```

Esercizio 3

Si scriva un programma in C che prende in input i seguenti valori: filein Comando Cstop Cecc dove:

- *filein: nome di un file leggibile.*
- *Comando: nome di un file eseguibile.*
- *Cstop, Cecc: singoli caratteri.*

Il processo iniziale (P0) deve creare un processo figlio (P1). P1 dovra' leggere il contenuto del file filein, e trasferirlo integralmente al processo padre P0. Il processo P0, una volta creato il processo figlio P1, dovrà leggere e stampare sullo standard output quanto inviatogli dal processo figlio P1, secondo le seguenti modalità:

- *Ogni carattere letto diverso da Cstop e da Cecc, viene stampato da P0 sullo standard output;*
- *Nel caso in cui P0 legga il carattere Cstop, dovrà semplicemente terminare forzatamente l'esecuzione di entrambi i processi;*
- *Nel caso in cui P0 legga il carattere Cecc, P0 dovrà interrompere l'esecuzione del figlio P1; P1 dal momento dell'interruzione in poi, passerà ad eseguire il comando Comando, e successivamente terminerà.*

Scegliere un comando semplice da eseguire, es. ls o pwd. Stampare a video i diversi comportamenti.

```c
#include "stdio.h"
#include "stdlib.h"
#include "signal.h"
#include "unistd.h"

char* comando;

void signal_handler(int s){
    printf("\nP1: Ricevuto il segnale.");
    execlp(comando, comando, NULL);
    exit(0);
}

int main(int argc, char** argv){
    printf("\nInizio del programma\n");
    comando = argv[2];
    char Cstop = argv[3][0];
    char Cecc = argv[4][0];
    int comm_pipe[2];
```

```
    pipe(comm_pipe);

    int pid = fork();
    if(pid==0){
        signal(SIGUSR1, signal_handler);
        close(comm_pipe[0]);
        printf("P1: Inizio a leggere da file.\n");
        FILE* filein = fopen(argv[1], "r");
        char c;
        do{
            c = fgetc(filein);
            write(comm_pipe[1], &c, sizeof(char));
        } while (c!=EOF);
        printf("P1: Finito di leggere, aspetto il segnale.\n");
        pause();
    }

    close(comm_pipe[1]);
    printf("P0: Inizio a leggere da pipe.\n");
    char c;
    printf("P0: ");
    do{
        read(comm_pipe[0], &c, sizeof(char));
        printf("%c",c);
        if(c==Cstop) kill(pid, SIGKILL), exit(0);
        if(c==Cecc) kill(pid, SIGUSR1), exit(0);
        if(c==EOF) printf("P0: Nessun carattere significativo ricevuto.\n"), exit(1);
    } while(1);

}
```

Esame del 06/03/2023 (GitHub)

Esercizio 1

Utilizzando opportuni comandi in concatenazione si eseguano le seguenti operazioni:

(a) *Utilizzando awk si scriva un comando che stampi una lista dei file presenti nella directory corrente mostrando solo nome e proprietario.*

```
ls -l | awk '{print $9" "$3}'
```

(b) *Si calcoli la dimensione occupata in totale dai file regolari con dimensione inferiore di 1024 byte nella directory corrente.*

```
ls -l | awk '{ if($1 ~ /^-/ && $5<1024) somma=somma+$5 } END {print somma}'
```

(c) *Dato un file "parole.txt" stampa solo le linee con più di 10 caratteri.*

```
awk --field-separator= '{if(NF>10) print $0}' parole.txt
```

(d) *Impostando una variabile d'ambiente LIST: perm, link, user, group, date (può anche essere inizializzata al di fuori del comando stesso) visualizzi il listato dei file nella directory corrente con il campo corrispondente*

```
export LIST="perm,link,user,group,date"
ls -l | awk -v list="$LIST" '{split(list, fields, ","); print $fields[1], $fields[2], $fields[3],
$fields[4], $fields[5]}'
```

Esercizio 2

Si realizzi uno script di shell BASH "`menu`", che implementi le seguenti funzioni accessibili da un menu:

(a) Aggiungi verifica - Chiede all'utente gli elementi: giorno, mese, anno, nome studente, voto. E inserirli in un file di nome "`verifica`". Il file deve essere creato sola la prima volta, chiamate successive, dovranno aggiungere nuove righe allo stesso file.
(b) Conta - Chiede all'utente il mese e lo studente, e conta il numero di prove effettuate nel mese per lo studente dato.
(c) Media - Chiede all'utente lo studente, e calcola la media dei voti delle verifiche date dallo studente.

Si rappresentino i mesi con una stringa di tre caratteri (gen, feb, mar, ecc.)

```
#!/bin/bash

scelta=0
while [ $scelta -ne 4 ];
do
    echo "Scegli un'opzione"
    echo "1) Aggiungi verifica"
    echo "2) Conta"
    echo "3) Media"
    echo "4) Uscita"
    read scelta
    case $scelta in
        1) echo "Inserisci il giorno: "
            read giorno
            echo "Inserisci il mese nel formato di 3 lettere (gen feb mar): "
            read mese
            echo "Inserisci l'anno: "
            read anno
            echo "Inserisci il nome dello studente: "
            read nome
            echo "Inserisci il voto: "
            read voto
```

```
            echo $giorno,$mese,$anno,$nome,$voto >> verifica
            ;;
        2) echo "Inserisci il nome dello studente: "
            read nome
            echo "Inserisci il mese in cui contare le prove: "
            read mese
            risultato=$(awk -F, -v mese="$mese" -v nome="$nome" 'BEGIN{somma=0} {if($2==mese && $4==nome)
somma=somma+1} END {print somma}' verifica)
            echo "In $mese ci sono $risultato prove per $nome"
            ;;
        3) echo "Inserisci il nome dello studente: "
            read nome
            somma=$(awk -F, -v nome="$nome" 'BEGIN{somma=0} {if($4==nome) somma=somma+$5} END {print somma}'
verifica)
            numero=$(awk -F, -v nome="$nome" 'BEGIN{somma=0}{if($4==nome) somma=somma+1} END {print somma}'
verifica)
            media=$((somma/numero))
            echo "La media è: $media"
            ;;
        4) echo "Uscita"
            ;;
        *) echo "Scelta non valida"
            ;;
    esac
done
```

Esercizio 3

Si scriva un programma in C che prende in input i seguenti valori: N N1 N2 C, dove:

- *N, N1, N2 sono interi positivi*
- *C è il nome di un file eseguibile (presente nel PATH)*

Il processo iniziale 'padre' (P0) deve creare 2 processi figli: P1 e P2, dopodiché:

- *Il figlio P1 deve aspettare N1 secondi e successivamente eseguire il comando C;*
- *Il figlio P2 dopo N2 secondi dalla sua creazione dovrà provocare la terminazione del processo fratello P1 e successivamente terminare; nel frattempo P2 deve periodicamente sincronizzarsi con il padre P0 (si assuma la frequenza di 1 segnale al secondo).*
- *Il padre P0, dopo aver creato i figli, si pone in attesa di segnali da P1: per ogni segnale ricevuto, dovrà stampare il proprio pid; al N-simo segnale ricevuto dovrà attendere la terminazione dei figli e successivamente terminare*

Scegliere un comando semplice da eseguire, es. ls o pwd. Stampare a video i diversi comportamenti.

```c
#include "stdio.h"
#include "stdlib.h"
#include "sys/wait.h"
#include "unistd.h"
#include "signal.h"

void signal_handler(int s){ ; }

int main (int argc, char** argv) {
    printf("\nINIZIO PROGRAMMA\n");
    int N = atoi(argv[1]);
    int N1 = atoi(argv[2]);
    int N2 = atoi(argv[3]);
    char* comando = argv[4];

    int pid1 = fork();
    if(pid1==0){
        printf("P1: Inizia l'esecuzione.\n");
        fflush(stdout);
        sleep(N1);
        printf("P1: Terminata l'attesa.\n");
        execlp(comando, comando, NULL);
        printf("Errore nell'esecuzione del comando.\n");
        return 0;
```

```
    }
    int pid2 = fork();
    if(pid2==0){
        printf("P2: Inizia l'esecuzione.\n");
        while(N2){
            kill(getppid(), SIGUSR1);
            N2--;
            printf("P2: Segnale USR1 inviato a P0.\n");
            sleep(1);
        }
        kill(pid1, SIGKILL);
        printf("P2: Segnale KILL inviato a P1.\nP2: Termino.\n");
        return 0;
    }
    signal(SIGUSR1, signal_handler);
    printf("P0: Inizia l'esecuzione.\n");
    while(N) {
        pause();
        N--;
    }
    printf("P0: Ricevuti tutti o segnali. Attendo P1 e P2.\n");
    while(wait(NULL)>0) ;

    printf("P0: Termino.\n");
    return 0;
}
```

Esame del 13/06/2023 (GitHub)

Esercizio 1

Utilizzando opportuni *comandi* in concatenazione si eseguano le seguenti operazioni:

A. Utilizzando awk si scriva un comando che stampi solo le linee di un file "parole.txt" che iniziano con la parola "LSO".

```
awk '{if( $0 ~ /^LSO/) print $0}' parole.txt
awk '/^LSO/ {print}' parole.txt
```

B. *Usando grep si stampino tutte le linee che terminano con "700".*

```
grep "700$" parole.txt
```

C. *Dato un file contenente i compleanni di un gruppo di amici, usando sed restituire le linee degli amici aventi il compleanno tra novembre e dicembre.*

```
sed -n '/..-\(11\|12\)-..../p' compleanni.txt
```

D. *Data una file contente nomi, cognome e data di nascita, usando awk restituire tutti i cognomi che iniziano con R.*

```
awk '{if ($2 ~ /^R/) print}' nomicognomi.txt
awk '$2 ~ /^R/' nomicognomi.txt
```

Esercizio 2

Dato un file di testo "paghe.txt" con almeno 6 righe di testo, scrivere uno script "stipendi" che inserisca il titolo "Sig.re" se si tratta di un uomo, e il titolo "Sig.ra" se si tratta di una donna, prima del nome. Calcolare e mostrare a video lo stipendio minimo, massimo e medio del personale, e aggiungere un bonus di x euro (dove x viene inserito dall'utente) allo stipendio minimo. Il file dovrà contenere i seguenti campi: nome, cognome, genere, stipendio, anno di assunzione.

```
#!/bin/bash

aggiungi_titolo(){
    if [ "$1" == "M" ]; then
        echo "Sig.re $2"
    elif [ "$1" == "F" ]; then
        echo "Sign.ra $2"
    else
        echo $2
    fi
}

stipendio_minimo=9999999
stipendio_massimo=0
media_stipendi=0
numero_dipendenti=0
while IFS=, read -r  nome cognome genere stipendio assunzione; do
    nome_completo=$(aggiungi_titolo "$genere" "$cognome")
    if [ "$stipendio" -lt "$stipendio_minimo" ]; then
            stipendio_minimo="$stipendio"
    fi
    if [ "$stipendio" -gt "$stipendio_massimo" ]; then
        stipendio_massimo="$stipendio";
    fi
    media_stipendi=$(( media_stipendi + stipendio ))
    numero_dipendenti=$(( numero_dipendenti + 1 ))
echo "$nome_completo, $nome, $genere, $stipendio, $assunzione"
done < paghe.txt
echo "Stipendio massimo: $stipendio_massimo"
echo "Stipendio minimo: $stipendio_minimo"
media_stipendi=$(( media_stipendi / numero_dipendenti ))
echo "Media stipendi: $media_stipendi"
```

```
read -p "Inserisci il bonus per lo stipendio minimo: " bonus
read -p "Si vuole modificare il file incrementando lo stipendio minimo? (y/n) " modifica

if [ "$modifica" == "y" ]; then
    sed -i "s/$stipendio_minimo/$((stipendio_minimo+bonus))/g" paghe.txt
else
     sed "s/$stipendio_minimo/$((stipendio_minimo+bonus))/g" paghe.txt
fi
```

Esercizio 3

Realizzare un programma C che, utilizzando le system call di UNIX, che prevede: esame F N C:

- *F rappresenta il nome assoluto di un file*
- *N rappresenta un intero*
- *C rappresenta un carattere*

Il processo iniziale Il processo iniziale P0 deve creare un numero due processi figli P1 e P2. P1 legge una parte del file F: in particolare, se L è la lunghezza del file F, il figlio dovrà leggere una frazione di L/N caratteri dal file F, e lo invia al figlio P2. Il processo P1 leggerà quindi una frazione di F allo scopo di calcolare il numero delle occorrenze del carattere C nella parte di file esaminata; al termine della scansione, P1 comunicherà al padre il numero delle occorrenze di C incontrate nella frazione di file assegnatagli. Il figlio P2 conferma al padre il numero di occorrenze di C del pezzo inviatogli. Il padre P0, una volta ottenuti i risultati da tutti i figli, stamperà il numero totale di occorrenze di C nel file F e terminerà. I diversi passaggi devono essere mostrati a video.

```
#include "stdio.h"
#include "stdlib.h"
#include "unistd.h"
#include "sys/wait.h"

int main(int argc, char** argv){


    int N = atoi(argv[2]);
    char C = argv[3][0];

    // Creazione pipe
    int pipe1[2], pipe2[2];
    if(pipe(pipe1) == -1 || pipe(pipe2) == -1) printf("Errore: impossibile creare le pipe\n"), return -1;

    // Creazione processo figlio 1
    int pid1 = fork();
    if(pid1 == 0){
        printf("P1 INIZIA\n");

        // Chiusura pipe inutilizzate
        close(pipe1[0]);
        close(pipe2[0]);

        // Apertura file
        FILE* file = fopen(argv[1], "r");

        printf("P1 CALCOLA LUNGHEZZA DA LEGGERE\n");
        fseek(file, 0, SEEK_END);
        int length = ftell(file);
        fseek(file, 0, SEEK_SET);
        int pieceLength = length / N;
        if(length % N != 0)
            pieceLength++;

        // Allocazione buffer
        char* buffer = malloc(pieceLength * sizeof(char));

        printf("P1 LEGGE IL PEZZO\n");
        fseek(file, 0, SEEK_SET);
        fread(buffer, sizeof(char), pieceLength, file);
        fclose(file);

        printf("P1 INVIA IL PEZZO\n");
        write(pipe1[1], buffer, pieceLength);
```

```
        printf("P1 CONTA LE OCCORRENZE DI %c\n", C);
        int occurrences = 0;
        for(int i = 0; i < pieceLength; i++)
            if(buffer[i] == C)
                occurrences++;
        printf("P1 HA TROVATO %d OCCORRENZE DI %c\n", occurrences, C);

        printf("P1 INVIA IL NUMERO OCCORRENZE A P0\n");
        write(pipe2[1], &occurrences, sizeof(int));
        free(buffer);

        printf("P1 TERMINA\n");
        close(pipe1[1]);
        return 0;
    }

    // Creazione processo figlio 2
    int pid2 = fork();
    if(pid2 == 0){
        printf("P2 INIZIA\n");

        // Chiusura pipe inutilizzate
        close(pipe1[1]);
        close(pipe2[0]);

        // Allocazione buffer
        char* buffer = malloc(1024 * sizeof(char));

        // Lettura pezzo
        printf("P2 LEGGE DALLA PIPE IL PEZZO\n");
        int readLength = read(pipe1[0], buffer, 1024);

        printf("P2 CALCOLA IL NUMERO OCCORRENZE DI %c\n", C);
        int occurrences = 0;
        for(int i = 0; i < readLength; i++)
            if(buffer[i] == C)
                occurrences++;
        printf("P2 HA TROVATO %d OCCORRENZE DI %c\n", occurrences, C);


        printf("P2 INVIA IL NUMERO OCCORRENZE A P0\n");
        write(pipe2[1], &occurrences, sizeof(int));
        free(buffer);

        printf("P2 TERMINA\n");
        close(pipe1[0]);
        close(pipe2[1]);
        return 0;
    }

    printf("P0 INIZIA\n");
    // Chiusura pipe inutilizzate
    close(pipe1[0]);
    close(pipe1[1]);
    close(pipe2[1]);

    printf("P0 ASPETTA CHE P1 E P2 TERMININO\n");
    waitpid(pid1, NULL, 0);
    waitpid(pid2, NULL, 0);

    printf("P0 LEGGE IL NUMERO OCCORRENZE DA P1\n");
    int occurrences1;
    read(pipe2[0], &occurrences1, sizeof(int));
    printf("P0 LEGGE IL NUMERO OCCORRENZE DA P2\n");
    int occurrences2;
    read(pipe2[0], &occurrences2, sizeof(int));
    printf("P1 E P2 HANNO TROVATO LO STESSO NUMERO DI %c? %s\n", C, occurrences1==occurrences2 ? "true" :
"false");

    return 0;
}
```

Esame del 13/06/2023 (GitHub)

Esercizio 1

Utilizzando opportuni *comandi* in concatenazione si eseguano le seguenti operazioni:

A. *Dato un file avente contenuti "nome cognome" restituire a video o in un altro file, i nomi e cognomi in maniera inversa e separati da virgola. Es. "alessandra rossi"-> "rossi, alessandra".*

```
awk '{ print $2", "$1 }' nomicognomiesercizio1.txt
```

B. *Usando awk stampare tutti i numeri di telefono presenti in un file avente struttura: nome, cognome, numero di telefono, indirizzo.*

```
awk -F , '{ print $3 }' nomicognomiesercizio1.txt
```

C. *Usando awk, dato un nome prestabilito, restituire il numero di telefono della persona indicata. Si usi lo stesso file usato al punto precedente.*

```
read -p "Inserisci il nome: "; $nome; awk -F , '{ if( $1 == "'$nome'") print "Il suo numero è: " $3 }' nomicognomiesercizio1.txt
```

D. *Data una file contente nomi, cognome e data di nascita, usando awk restituire tutti i cognomi che iniziano con R e termina con E.*

```
awk '$2 ~ /^R.*e,/' nomicognomiesercizio1.txt
```

Esercizio 2

Dato un file di testo "paghe.txt" con almeno 6 righe di testo, scrivere uno script "stipendi" che inserisca il titolo "Sig.re" se si tratta di un uomo, e il titolo "Sig.ra" se si tratta di una donna, prima del nome. Calcolare e mostrare a video lo stipendio minimo, massimo e medio del personale, e aggiungere un bonus di x euro (dove x viene inserito dall'utente) allo stipendio minimo. Il file dovrà contenere i seguenti campi: nome, cognome, genere, stipendio, anno di assunzione.

```
#!/bin/bash

aggiungi_titolo(){
    if [ "$1" == "M" ]; then
        echo "Sig.re $2"
    elif [ "$1" == "F" ]; then
        echo "Sign.ra $2"
    else
        echo $2
    fi
}

stipendio_minimo=9999999
stipendio_massimo=0
media_stipendi=0
numero_dipendenti=0
while IFS=, read -r  nome cognome genere stipendio assunzione; do
    nome_completo=$(aggiungi_titolo "$genere" "$cognome")
    if [ "$stipendio" -lt "$stipendio_minimo" ]; then
            stipendio_minimo="$stipendio"
    fi
    if [ "$stipendio" -gt "$stipendio_massimo" ]; then
        stipendio_massimo="$stipendio";
    fi
    media_stipendi=$(( media_stipendi + stipendio ))
    numero_dipendenti=$(( numero_dipendenti + 1 ))
echo "$nome_completo, $nome, $genere, $stipendio, $assunzione"
done < paghe.txt
echo "Stipendio massimo: $stipendio_massimo"
echo "Stipendio minimo: $stipendio_minimo"
media_stipendi=$(( media_stipendi / numero_dipendenti ))
echo "Media stipendi: $media_stipendi"
```

```
read -p "Inserisci il bonus per lo stipendio minimo: " bonus
read -p "Si vuole modificare il file incrementando lo stipendio minimo? (y/n) " modifica

if [ "$modifica" == "y" ]; then
    sed -i "s/$stipendio_minimo/$((stipendio_minimo+bonus))/g" paghe.txt
else
     sed "s/$stipendio_minimo/$((stipendio_minimo+bonus))/g" paghe.txt
fi
```

Esercizio 3

Si realizzi un programma C il cui processo padre P0 dia il via alla generazione di n processi in gerarchia lineare. Dove n è un numero intero passato come argomento al programma. Cioè, P0 genera P1, P1 genera P2, ..., Pn genera Pn+1. Il P0 deve prendere in input una sequenza di N comandi (per semplicità, senza argomenti e senza opzioni). Ogni N processo deve eseguire il rispettivo N. L'applicazione termina quando l'ultimo processo ha terminato.

```
#include "stdio.h"
#include "stdlib.h"
#include "unistd.h"
#include "sys/wait.h"


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

    printf("\nEsecuzione di %s\n\n", argv[0]);

    for(int i = 1; i < argc; i++) {
        int pid = fork();
        if(pid == 0) {
            execlp(argv[i], argv[i], NULL);
            printf("Errore nell'esecuzione di %s\n", argv[i]);
            exit(1);
        }
    }

    while (wait(NULL) > 0);


    printf("Esecuzione terminata\n");
    return 0;
}
```

Esame del 13/07/2023 (GitHub)

Esercizio 1

Utilizzando opportuni comandi in concatenazione si eseguano le seguenti operazioni:

(a) *Utilizzando sed sostituire tutte le occorrenze (case insensitive) di "LSO" con "LSO-2023" in un file di nome "esami_sostenuti.txt".*

```
sed -i 's/LSO/LSO-2023/ig' esami_sostenuti.txt
```

(b) *Usando awk stampare tutte il numero delle linee del "esami_sostenuti.txt" che contiene il pattern "LSO"*

```
awk 'BEGIN{somma=0} { if($0 ~ /LSO/) somma++ } END{print somma}' esami_sostenuti.txt
```

(c) *Usando awk, stampare la somma della dimensione dei file della directory corrente*

```
ls -l | awk 'BEGIN{somma=0} {if($1 ~ /^-/) somma+=$5} END{print somma}'
```

(d) *Dato un file avente contenuti "nome- cognome" restituire a video o in un altro file, i nomi e cognomi in maniera inversa e con nome proprio abbreviato. Es. "alessandra rossi"-> "rossi, a."*

```
awk -F'-' '{print $2", "$1}' nomicognomies1.txt |  sed -E 's/, ([^ ])([^ ])*/, \1./'
```

Esercizio 2

Scrivere uno script BASH che confronta il contenuto di due directory non vuote, e per ogni elemento differente in una directory, questo viene creato nell'altra. Stampare il contenuto di ogni directory, la dimensione di ogni directory, e stampare la lista di file/directory creati.

```bash
#!/bin/bash

# Verifica se sono state fornite due directory come argomenti
if [ "$#" -ne 2 ]; then
    echo "Usage: $0 <directory1> <directory2>"
    exit 1
fi
dir1="$1"
dir2="$2"
lista_copiati=()

# Funzione per confrontare il contenuto delle directory e copiare i file mancanti
compare_and_copy() {
    local source_dir="$1"
    local dest_dir="$2"

    # Itera su tutti gli elementi nella directory di origine
    for item in "$source_dir"/*; do
        base_name=$(basename "$item")

        # Verifica se l'elemento esiste nella directory di destinazione
        if [ ! -e "$dest_dir/$base_name" ]; then
            # Copia l'elemento mancante nella directory di destinazione
            cp -r "$item" "$dest_dir/"
            # Aggiungi il file alla lista dei file copiati
            lista_copiati+=("$base_name")
        fi
    done
}

# Esegue la funzione per entrambe le directory
compare_and_copy "$dir1" "$dir2"
compare_and_copy "$dir2" "$dir1"

# Stampa il contenuto di ogni directory
echo -e "\nContenuto di $dir1:"
ls -l "$dir1"

echo -e "\nContenuto di $dir2:"
```

```
ls -l "$dir2"

# Stampa le dimensioni delle directory
echo -e "\nDimensione di $dir1:"
ls -l | awk -v dir1="$dir1" '$9 ~ dir1 {print $5}'

echo -e "\nDimensione di $dir2:"
ls -l | awk -v dir2="$dir2" '$9 ~ dir2 {print $5}'

# Stampa i file copiati
echo -e "\nFile copiati:"
for file in "${lista_copiati[@]}"; do
    echo "$file"
done
```

Esercizio 3

Realizzare un programma C il cui processo iniziale P0 prende in input un numero N. Il processo P0 crea una gerarchia di processi di profondità N+1 (figlio P1- nipote P2- bisnipote P3- ....-PN+1). A partire dall'ultimo generato, ogni processo della gerarchia avrà il seguente comportamento:

- *se il suo pid è pari, esegue un comando ls*
- *se il pid è dispari, termina ed invia un segnale al padre P0*

Il P0 raccoglie lo stato di terminazione di tutti i figli, li stampa a video, e termina la propria esecuzione.

Tutte le operazioni rilevanti devono essere stampate a video.

```
#include "stdio.h"
#include "unistd.h"
#include "signal.h"
#include "stdlib.h"
#include "sys/wait.h"

void signal_handler(int signal){
    printf("Padre: Ricevuto segnale da figlio con pid dispari.\n");
}

int main(int argc, char** argv){

    int N = atoi(argv[1]);

    signal(SIGUSR1, signal_handler);

    for(int i=0; i<N; i++){
        printf("Nascita figlio %d-esimo.\n", i);
        int pid = fork();
        if(pid==0){
            if(getpid()%2==0){
                execlp("ls", "ls", NULL);
                printf("Errore nell'esecuzione del comando.\n");
                exit(1);
            }else{
                printf("Inviato segnale al padre.\n");
                kill(getppid(), SIGUSR1);
                return 0;
            }
        }
    }

    // Raccogli e stampa lo stato di terminazione dei figli
    int status;
    for(int i=0; i<N; i++){
        wait(&status);
        printf("Figlio %d-esimo terminato con stato %d.\n", i, status);
    }

    return 0;
}
```

Esame del 13/07/2023 (GitHub)

Esercizio 1

Utilizzando opportuni comandi in concatenazione si eseguano le seguenti operazioni:

(a) *Dato un file avente contenuti "nome- cognome" restituire a video o in un altro file, i nomi e cognomi in maniera inversa e con nome proprio abbreviato. Es. "alessandra rossi"-> "rossi, a."*

```
awk -F'-' '{print $2", "$1}' nomicognomies1.txt |  sed -E 's/, ([^ ])([^ ])*/, \1./'
```

(b) *Utilizzando sed sostituire tutte le occorrenze (case sensitive) di "LSO" con "LSO-2023" in un file di nome "esami_sostenuti.txt".*

```
sed -i 's/LSO/LSO-2023/g' esami_sostenuti.txt
```

(c) *Usando awk stampare tutte il numero delle linee del "esami_sostenuti.txt" che non contiene il pattern "LSO"*

```
awk 'BEGIN{somma=0} { if($0 ~ /LSO/) somma++ } END{print NR-somma}' esami_sostenuti.txt
```

(d) Usando awk, stampare i nomi dei file aventi dimensione inferiore della directory corrente

```
size=$(ls -l | awk 'BEGIN{somma=0} {if($1 ~ /^-/) somma+=$5} END{print somma}') ;
ls -l | awk -v size=$size '{if($5<size) print $9}'
```

Esercizio 2

Scrivere uno script BASH che confronta il contenuto di due directory non vuote, e per ogni elemento differente in una directory, questo viene creato nell'altra. Stampare il contenuto di ogni directory, la dimensione di ogni directory, e stampare la lista di file/directory creati.

```bash
#!/bin/bash

# Verifica se sono state fornite due directory come argomenti
if [ "$#" -ne 2 ]; then
    echo "Usage: $0 <directory1> <directory2>"
    exit 1
fi
dir1="$1"
dir2="$2"
lista_copiati=()

# Funzione per confrontare il contenuto delle directory e copiare i file mancanti
compare_and_copy() {
    local source_dir="$1"
    local dest_dir="$2"

    # Itera su tutti gli elementi nella directory di origine
    for item in "$source_dir"/*; do
        base_name=$(basename "$item")

        # Verifica se l'elemento esiste nella directory di destinazione
        if [ ! -e "$dest_dir/$base_name" ]; then
            # Copia l'elemento mancante nella directory di destinazione
            cp -r "$item" "$dest_dir/"
            # Aggiungi il file alla lista dei file copiati
            lista_copiati+=("$base_name")
        fi
    done
}

# Esegue la funzione per entrambe le directory
compare_and_copy "$dir1" "$dir2"
compare_and_copy "$dir2" "$dir1"

# Stampa il contenuto di ogni directory
echo -e "\nContenuto di $dir1:"
ls -l "$dir1"
```

```
echo -e "\nContenuto di $dir2:"
ls -l "$dir2"

# Stampa le dimensioni delle directory
echo -e "\nDimensione di $dir1:"
ls -l | awk -v dir1="$dir1" '$9 ~ dir1 {print $5}'

echo -e "\nDimensione di $dir2:"
ls -l | awk -v dir2="$dir2" '$9 ~ dir2 {print $5}'

# Stampa i file copiati
echo -e "\nFile copiati:"
for file in "${lista_copiati[@]}"; do
    echo "$file"
done
```

Esercizio 3

Si scriva un programma in C dove il processo padre P0 crea N processi figli (P1, P2...PN) con N dato in input dall'utente. Tutti i processi figli una volta creati restano in attesa di un segnale dal padre, quando ricevuto il segnale eseguono un comando "ls". Il processo P0 attiva il comportamento in base al proprio pid.

- *Se il ppid è pari attiva i figli con pid pari, e termina subito quelli con pid dispari*
- *Se il ppid è dispari attiva i figli con pid dispari, e termina subito quelli con pid pari*

Il P0 raccoglie lo stato di terminazione di tutti i figli, li stampa a video, e termina la propria esecuzione.

Tutte le operazioni rilevanti devono essere stampate a video.

```
#include "unistd.h"
#include "stdlib.h"
#include "stdio.h"
#include "signal.h"
#include "stdbool.h"

void signal_handler(int s){
    printf("Svegliato un %s\n", (getpid()%2) ? "dispari" : "pari");
}

int main(int argc, char** argv){

    int N = atoi(argv[1]);
    int* pids = (int*)malloc(N*sizeof(int));

    for(int i=0; i<N; i++){
        pids[i] = fork();
        if(pids[i]==0){
            signal(SIGUSR1, signal_handler);
            pause();
            execlp("ls", "ls", NULL);
            exit(0);
        }
    }

    bool dispari = (getppid()%2) ? true : false;
    for(int i=0; i<N; i++){
        if(dispari){
            if(i%2) kill(pids[i], SIGUSR1);
            else kill(pids[i], SIGKILL);
        } else {
            if(i%2) kill(pids[i], SIGKILL);
            else kill(pids[i], SIGUSR1);
        }
    }

    sleep(1);
    return 0;
}
```