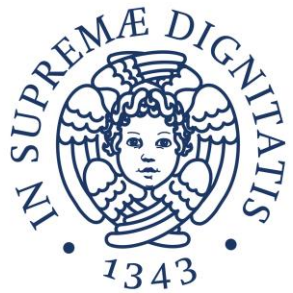

LABORATORIO DI SISTEMI OPERATIVI

Corso di Laurea in Ingegneria Informatica

A.A. 2023/2024

Ing. Maurizio Palmieri


maurizio.palmieri@unipi.it

Esercitazione 11

Pilotare Applicazioni

- Primitiva dup2
- Redirezione STDIN e STDOUT
- Pilotare Applicazioni con CLI
- Esercizi

STDIN STDOUT e STDERR

- STDIN, STDOUT e STDERR sono descrittori di defaults.
  - Vengono generati automaticamente al momento dell'esecuzione del programma
    - La loro apertura e chiusura viene gestita dal sistema operativo
  - Possono essere acceduti con le seguenti MACRO
    - definite in unistd.h
  - STDIN_FILENO
    - Macro che rappresenta il file descriptor dello standard input
  - STDOUT_FILENO
    - Macro che rappresenta il file descriptor dello standard output
  - STDERR_FILENO
    - Macro che rappresenta il file descriptor dello standard error

Comando dup2

Funzione per duplicare un file descriptor

- int **dup2**(int *target*, int *newfd*);
  - target è il file descriptor da duplicare
  - *newfd* è il file descriptor dove verrà messa la copia di *target*
  - il ritorno è negativo se c'è stato un errore
  - La funzione **chiude il file descriptor *newfd*** (se precedentemente aperto) prima di duplicare *target*

Redirezione standard file descriptors

- Si possono combinare le funzioni *pipe* e *dup2* per redirigere il flusso dei dati dagli standard verso altri fd

```
int main() {
    int pipe_fd[2];
    pid_t pid;

    pipe(pipe_fd);

    pid = fork();

    if (pid == 0){//figlio
            //chiudo estremità non usata
            close(pipe_fd[0]);

            //redirigo lo stdout verso la pipe
            dup2(pipe_fd[1], STDOUT_FILENO);

            execl("/bin/ls", "ls", "-l", NULL);
    }
...
```

Redirezione standard file descriptors

- Continuazione della slide precedente

```
...
    if (pid > 0){
            char buffer[1024];
            int nread = -1;
            int index = 0;

            //chiudo estremità non usata
            close(pipe_fd[1]);

            //approccio generale per la lettura
            while(nread != 0)
            {
                    nread=read(pipe_fd[0], &buffer[index],sizeof(buffer));
                    buffer[index+nread] = '\0';
                    index+=nread;
            }
            printf("Il padre ha letto:%s\n", buffer);
    }
    return 0;
}
```

Redirezione di un'applicazione

- Avete un applicazione che non potete modificare
  - non avete il sorgente
  - non conoscete il linguaggio usato
  - ....
- L'applicazione vi permette di inserire degli input da terminale(STDIN) e vi restituisce un output sul terminale (STDOUT)
- Volete "pilotare" l'applicazione con un programma C
  - create 2 pipe
  - fork
  - redirigete gli stdin e stdout del figlio
  - trasformate il figlio nell'applicazione
  - il padre pilota il figlio seguendo:
    - le esigenze del programmatore
    - i vincoli dell'applicazione

Esecuzione applicazione con redirezione

```
int FtS[2]; // father 2 son
int StF[2]; // son 2 father
pid_t pid;

pipe(FtS);
pipe(StF);
pid = fork();
if(pid == 0){//figlio
        // chiudo le estremità non utilizzate
        close(FtS[1]);
        close(StF[0]);

        // duplico FtS[0] e lo metto al posto di stdin e chiudo la copia extra
        dup2(FtS[0], STDIN_FILENO);
        close(FtS[0]);

        // duplico StF[1] e lo metto al posto di stdout e chiudo la copia extra
        dup2(StF[1], STDOUT_FILENO);
        close(StF[1]);

        execl(<path_applicazione>,<nome_applicazione>,NULL);
```

Esecuzione applicazione con redirezione

```
else{ //padre
     // un buffer per ricevere e uno per inviare
     char sendbuffer[1024];
     char receivebuffer[1024];

     //intero per gestire il ritorno della read
     int nread;

     //chiudo le estremità non utilizzate delle pipe
     close(FtS[0]);
     close(StF[1]);

     //preparo la stringa da mandare al figlio e la mando
      \n è l'invio(ritorno carrello)) ed è fondamentale
     sprintf(sendbuffer, "<formato stringa Applicazione>\n");
     write(FtS[1],sendbuffer,strlen(sendbuffer));

     //ricevo la risposta del figlio
     nread=read(StF[0], receivebuffer, sizeof(receivebuffer)-1);
     receivebuffer[nread] = '\0';
     printf(<risposta dell' Applicazione>);
}
```

ESERCIZI

Esercizio 1

Creare un programma in cui si redirige lo STDOUT del padre verso il figlio

- il main chiama la funzione pipe e poi chiama la fork
- Il padre usa la printf per mostrare il messaggio a video:
  - "No, io sono tuo padre"
- Il padre redirige lo STDOUT verso il pipe creato in precedenza
- Il padre usa la printf per mostrare il messaggio a video:
  - "<PID del figlio>, io sono tuo padre"
  - Cosa viene mostrato a video?

- Il figlio salva in un file "memoria.txt" quello che legge dal padre
  - Cercate con man il flag per creare un file con la open

Esercizio 2

Data la “applicazione” che trovate sul gruppo teams del corso, create un programma che la “piloti” nel seguente modo:

- Accetta un numero naturale *n* come argomento
- Interagisce 4 volte con l’applicazione nel seguente modo
  - invia *n* all’applicazione
  - stampa la risposta dell’applicazione
  - aggiorna *n* con il valore restituito dall’applicazione

- **Variante**
- Invece dell’applicazione usate il comando bc
  - man bc oppure bc -h
- per ottenere lo stesso risultato