04.04.2019

Curs 6 – S.O.

Pipe (anonim pe Win)

Capetele pipe-urilor trebuie închise imediat ce nu mai sunt necesare

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

int main (int argc , char ** argv ){
        int a[4] = {1, 2, 3, 4};
        int p[2];
        pipe(p);
        if (fork() == 0){

            close(p[0]);
              a[2] += a[3];
           write (p[1], &a[2], sizeof(int));
           close (p[1]);
           exit(0);
      }
      close(p[1]);
      a[0] += a[1];
      read ( p[0], &a[2], sizeof(int));
      close (p[0]);
      wait (0);
     a[0] += a[2];
     printf ("%d \n", a[0]);
     return 0;
}
```

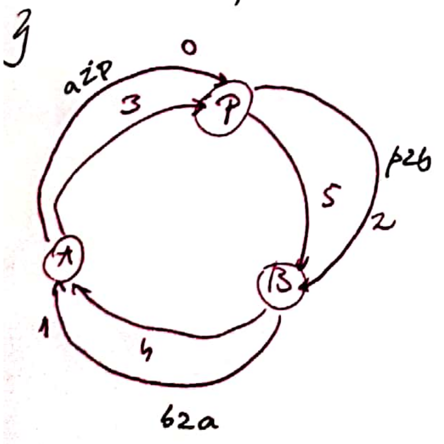
default: read ~~& write~~ nu citește
dacă pipe-ul e gol

→ odată ce încep să apară date, cum știe cât să citească

→ read e programat să ~~citea~~ citească cât există în pipe, dacă cerem 1000 de caractere, cel mai probabil nu am citi toate 1000 odată.

→ dacă pipe-ul e gol, read așteaptă până când ~~m~~ vin date sau nu mai are cine să scrie

→ read / write returnează câți octeți ~~corect~~ se citesc din ceea ce s-a cerut.



Hound-count.c

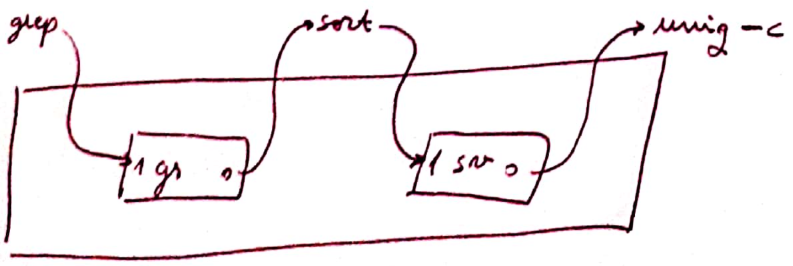
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main(int argc, char** argv){

    int p2b[2], b2a[2], a2p[2], n;

    pipe(p2b); pipe(b2a); pipe(a2p);

    if(fork() ==0 ){      //A
        close(p2b[0]); close(p2b[1]); close(b2a[1]); close(a2p[0]);
     while(1){
       if(read(b2a[0], &n, sizeof(int)) <= 0) break; if(n<=0) break;
         n--; printf("A: %d → %d\n", n+1, n);
         write(a2p[1], &n, sizeof(int));
     }
     close(b2a[0]); close(a2p[1]);
     exit(0);
   }

   if(fork() ==0){   //B

      close(a2p[0]); close(a2p[1]); close(p2b[1]); close(b2a[0]);
      while(1){
         if(read(p2b[0], &n, sizeof(int))<=0&& n<=0) break;
         n--; printf("B: %d → %d \n", n+1, n);
         write(b2a[1], &n, sizeof(int));
      }
      close(p2b[0]); close(b2a[1]);
      exit(0);
   }

  close(b2a[0]); close(b2a[1]); close(a2p[1]); close(p2b[0]);
  n=15;
 write(p2b[1], &n, sizeof(int));
  while(1){
      if(read(a2p[0], &n, sizeof(int))<=0 && n<=0) break;
      n--; printf("P: %d → %d \n", n+1, n);
      write(p2b[1], &n, sizeof(int));
  }
  close(a2p[0]); close(p2b[1]); wait(0); wait(0);
  return 0;
}
```

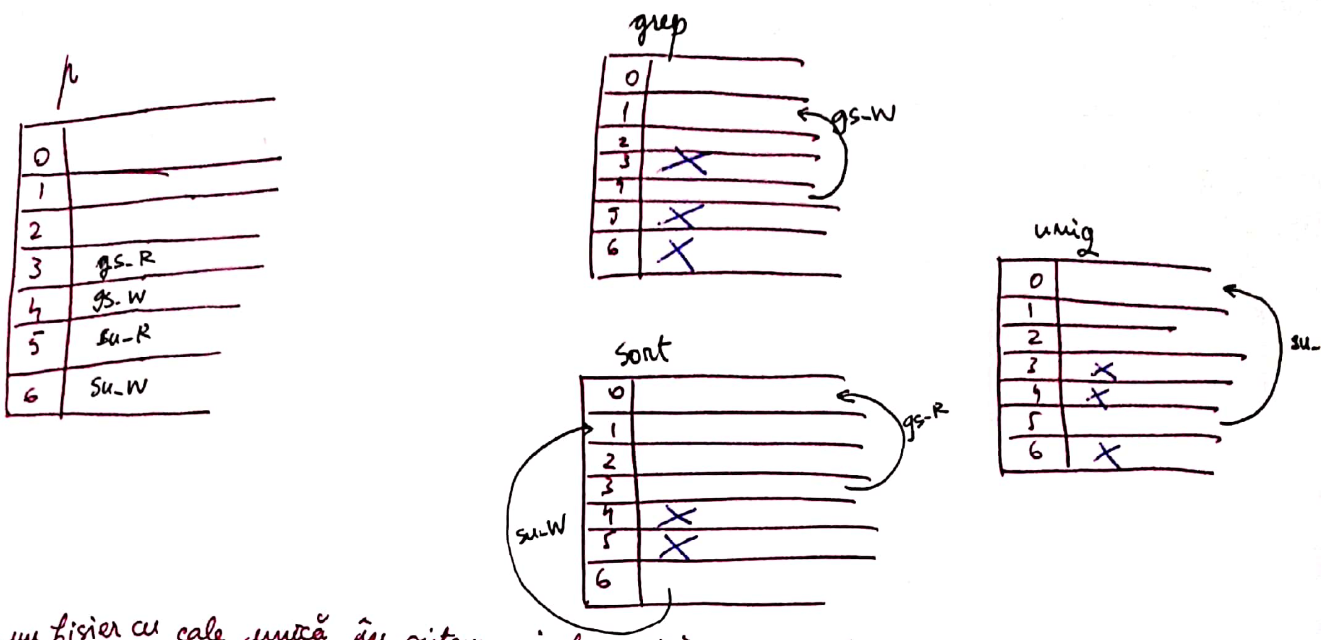
grep "an/gr211" /etc/passwd | sort | uniq -c
gs su

dup2 (p[0], 0) //→ va lua ce găsește la p[0] și va pune la 0 (doar în Procesul Curent)

pipe-chain.c

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


int main (int argc, char **argv){
  int gs[2], su[2]; pipe(gs); pipe(su);
  if(fork() ==0){ close (gs[0]); close (su[0]); close (su[1]); dup2 (gs[1], 1);
        execlp ( "grep", "grep", "/an1/ ", "/etc/passwd", NULL);
        ~~exit(0)~~
        exit(1);


  }
  if (fork() ==0) { close (gs[1]); close(su[0]); dup2 (gs[0], 0); dup2 ( su[1], 1);
       execlp ("sort", "sort", NULL);
       exit(1);


  }
  if (fork() ==0) { close (gs[0]); close(gs[1]); close (su[1]); dup2 (su[0], 0);
      execlp ( "uniq", "uniq", "-c", NULL);
      exit(1);


  } close(gs[0]); close (gs[1]); close (su[0]); close (su[1]);
  wait(0); wait (0); wait(0);
  return 0;
}
```





Fifo — un fișier cu cale unică în sistem și lucrează ca un pipe

mkfifo nume

la fifo, open are mecanism special ca read/write ♡