**SEMAFORI** strumento di sincronizzazione fra processi (int)

semget(ftok(file, key),numero\_semafori, flag)  
serve per creare o caricare un set di semafori con n\_semafori, nei flag si può specificare create, execl e i permessi.

semctl(id\_sem, flag\_remove, indice) per rimuovere un semaforo dal set

semctl(id\_sem, flag\_setval, indice, new\_valore) per modificare il valore di un semaforo nel set

semop(id\_sem, struttura\_operazione, numero\_operazioni) per effettuare operazioni sul set di semafori

**MEMORIA CONDIVISA** tra processi per lo scambio di informazioni

shmget(ftok(file, key),size, flag)

serve per creare o caricare un memoria, nei flag si può specificare create, execl e i permessi

shmctl(id\_shm, flag\_remove, indice) per rimuovere una memoria condivisa

shmat(shm\_id, NULL,0) per attaccare il segmento di memoria allo spazio di indirizzamento del processo, NULL indirizzo automatico, 0 lettura e scrittura

shmdt(id\_shm) sgancia la memoria dal processo chiamante

**CONTROLLO PROCESSI**

getpid() ritorna il pid del chiamante

getppid() ritorna il pid del padre

fork() crea un nuovo processo detto figlio

exit(status) termina un processo, status rappresenta lo status di terminazione

wait(&status) attende la terminazione del figlio e restituisce lo status in &status

execve(pathname, argv, envp) carica un nuovo programma nella memoria del processo e viene cancellato il testo del programma precedente

waitpid(pid, status, options) permette di attendere uno specifico figlio se pid > 0

system(command) permette di eseguire un comando shell

**SEGNALI** segnali a processi che qualcosa è accaduto

signal(signum,\*handler) assegna al segnale sig la funzione handler ritorna il vecchio puntatore al vecchio handler

kill(pid, signum) killa un processo, pid identifica uno o più processi e invia il segnale sig

sigaction(signum, struct new\_handler, struct old\_handler) come signal() ma con struct

sigprocmask(int how, \*new\_set, \*old\_set) modifica la mask di un processo

**PIPE** strumento di comunicazione tra processi (buffer)

int pipe(int filedes[2]) crea un pipe con l’array di descrittori per scrittura e lettura

Tra padre e figlio prima si crea il pipe() poi si fa la fork()

File \*popen(char \*command, char \*mode) apre un pipe, fa la fork() il figlio esegue una execs(), apre una shell che a sua volta crea un figlio che esegue il command. mode può essere r o w per il processo chiamante

int pclose(\*FILE stream) chiude il pipe e restituisce lo stato del processo figlio

**FIFO**

int mkfifo(char \*pathname, mode\_t mode) crea un fifo nel pathname e con i permessi mode

**CODA DI MESSAGGI**

int msgget(ftok(file, key), flag) crea una nuova coda, con i vari flag es. IPC\_CREAT o IPC\_EXEC

int msgctl(key, flag,0) per cancellare una coda con flag RMID

int msgsnd(msqid,&message,size,flag) per inviare un messaggio nella coda msqid. Il messaggio è contenuto nella struttura message

int msgsnd(msqid,&message,maxmsgsize,type,flag) per ricevere un messaggio dalla coda msqid, struttura message con size maxmsgsize, tipo type, flag.