

## PROCESSI

SYSTEM CALL	pid_t fork(void)
DESCRIZIONE	Creare un nuovo processo (child) invocando la duplicazione del processo chiamante (parent).
PARAMETRI	/
OUTPUT	<ul style="list-style-type: none"> <li>Nel parent: pid del figlio in caso di successo, -1 altrimenti</li> <li>Nel child: 0</li> </ul>

SYSTEM CALL	pid_t wait(int *status)
DESCRIZIONE	<p>Richiede la permanenza nello stato di blocco finché non si ha la terminazione di un generico processo figlio. Se un child ha già terminato, vuol dire che l'evento richiesto è già accaduto e non c'è bisogno di un'attesa.</p> <p>• int * status → puntatore alla cella di memoria in cui il Kernel registrerà il codice di terminazione del processo figlio nei sei bit meno significativi; gli altri bit vengono sfruttati dal Kernel per comunicare altre informazioni riguardanti la terminazione del processo figlio (ad esempio per specificare che il byte relativo al codice di terminazione è senza significato nel caso in cui il processo figlio venga interrotto con un ctrl+C e non con l'invocazione di una exit() ).</p>
PARAMETRI	
OUTPUT	-1 in caso di fallimento

	<b>SYSTEM CALL</b>	pid_t waitpid(pid_t pid, int *status, int options)
	<b>DESCRIZIONE</b>	Richiede la permanenza nello stato di blocco finché non si ha la terminazione di uno specifico processo figlio. Se quel child ha già terminato, non c'è bisogno di un'attesa.
	<b>PARAMETRI</b>	<ul style="list-style-type: none"> <li>• pid_t pid → process id del processo figlio di cui si attende la terminazione.</li> <li>• int* status</li> <li>• int options → impostazioni sull'esecuzione della funzione;</li> </ul> <p>due delle tante possibilità sono:</p> <ol style="list-style-type: none"> <li>1) impostazioni di default;</li> <li>2) richiesta del codice di terminazione senza attendere mai in attesa (se il processo figlio deve ancora terminare, si ha un errore).</li> </ol>
	<b>OUTPUT</b>	-1 in caso di fallimento

	<b>FUNZIONE</b>	pid_t getpid(void)
	<b>DESCRIZIONE</b>	Chiede al Kernel di restituire il pid del processo corrente.
	<b>PARAMETRI</b>	/

	<b>FUNZIONE</b>	pid_t getppid(void)
	<b>DESCRIZIONE</b>	Chiede al Kernel di restituire il pid del processo padre.
	<b>PARAMETRI</b>	/

"STRINGA IMMUTABILE"	
FUNZIONE	<code>int exec(const char *path, const char *arg, ..., NULL)</code>
DESCRIZIONE	Richiede che il programma corrente sia sostituito con un altro programma. • <code>const char *path</code> → nome del programma che deve essere attivato; dovrebbe contenere tutto il path, ovvero il percorso assoluto (e.g. <code>/home/francesco/.../nome-programma</code> ). L'unica variante possibile è quella in cui la stringa è nella forma "nome-programma"; in tal caso, la posizione del programma nel file system è identificata relativamente al valore della variabile d'ambiente Process Working Directory (PWD): il percorso effettivo si ottiene mediante la stringa <code>PWD/nome-programma</code> .
PARAMETRI	• <code>const char *arg, ..., NULL</code> → sequenza composta da un numero n arbitrario di stringhe; le prime $n-1$ sono quelle da passare come parametro al main del programma che deve essere attivato.
OUTPUT	-1 in caso di fallimento

FUNZIONE	<code>int execvp(const char *file, const char *arg, ..., NULL)</code>
DESCRIZIONE	Richiede che il programma corrente sia sostituito con un altro programma.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>const char *file</code> → nome del programma che deve essere attivato; è della forma "nome-programma". Tuttavia, potrebbero esserci più programmi denominati con "nome-programma" in più directory diverse; qui entra in gioco un'altra variabile d'ambiente, <code>PATH</code>, che contiene una collezione di valori (per esempio, nella forma <code>/home/francesco/bin:/usr/local/bin:...:/usr/lib/mit/sbin</code>): la funzione chiede al Kernel di lanciare un programma che si chiama "nome-programma" nel directory <code>/home/francesco/bin</code>. Se ciò non è possibile, la funzione chiede di attivare il programma nella directory <code>/usr/local/bin</code>, e così via.</li> <li>• <code>const char *arg, ..., NULL</code></li> </ul>
OUTPUT	-1 in caso di fallimento

FUNZIONE	<code>int execvpe(const char *file, char *const argv[], char *const envp[])</code>
DESCRIZIONE	Richiede che il programma corrente sia sostituito con un altro programma.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>const char *file</code></li> <li>• <code>char *const argv[]</code> → array di puntatori a carattere, utilizzato per comunicare al Kernel i parametri da passare al main del programma che deve essere attivato.</li> <li>• <code>char *const envp[]</code> → array di puntatori a carattere, utilizzato per comunicare al Kernel quali sono le stringhe che dovranno essere collocate nella zona bassa dello stack del nuovo address space per rappresentare le informazioni ambientali.</li> </ul>
OUTPUT	-1 in caso di fallimento

FUNZIONE	<code>int execle(const char *path, const char *arg, ..., char *const envp[])</code>
DESCRIZIONE	Richiede che il programma corrente sia sostituito con un altro programma.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>const char *path</code></li> <li>• <code>const char *arg, ...</code></li> <li>• <code>char *const envp[]</code></li> </ul>
OUTPUT	-1 in caso di fallimento

FUNZIONE	<code>int execv(const char *path, char *const argv[])</code>
DESCRIZIONE	Richiede che il programma corrente sia sostituito con un altro programma.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>const char *path</code></li> <li>• <code>char *const argv[]</code></li> </ul>
OUTPUT	-1 in caso di fallimento

FUNZIONE	<code>int execvp(const char *file, char *const argv[])</code>
DESCRIZIONE	Richiede che il programma corrente sia sostituito con un altro programma.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>const char *file</code></li> <li>• <code>char *const argv[]</code></li> </ul>
OUTPUT	-1 in caso di fallimento

SYSTEM CALL	<code>int execve(const char *path, char *const argv[], char *const envp[])</code>
DESCRIZIONE	Richiede che il programma corrente sia sostituito con un altro programma.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>const char *path</code></li> <li>• <code>char *const argv[]</code></li> <li>• <code>char *const envp[]</code></li> </ul>
OUTPUT	-1 in caso di fallimento

FUNZIONE	<code>char *getenv(char *name)</code>
DESCRIZIONE	Preleva il valore di una delle variabili d'ambiente attualmente presenti nell'ambiente (operazione molto semplice dato che queste variabili sono nella forma "nome=valore" oppure "nome=valore1;valore2;...;valoren").
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>char *name</code> → nome della variabile d'ambiente</li> </ul>
OUTPUT	Un puntatore all'area di memoria in cui viene inserito il valore della variabile se il nome della variabile fornito in input esiste; NULL altrimenti

FUNZIONE	<code>int putenv(char *string)</code>
DESCRIZIONE	Inserisce una nuova variabile all'interno dell'ambiente.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>char *string</code> → puntatore a carattere, che punta a una stringa della forma "nome=valore" oppure "nome=valore1;valore2;...;valoren".</li> </ul>
OUTPUT	0 in caso di successo, un valore diverso altrimenti (si ha un fallimento se per esempio la variabile era già presente nell'ambiente)



FUNZIONE	int setenv (char *name, char *value, int overwrite)
DESCRIZIONE	Setta il valore di una variabile d'ambiente. • char *name → nome della variabile d'ambiente • char *value → valore da assegnare alla variabile
PARAMETRI	• int overwrite → flag che può offrire la possibilità di andare in overwrite nel caso in cui la variabile esiste già nell'ambiente. L'overwrite consiste nell'eliminazione della vecchia variabile che viene rimpiazzata da quella nuova col valore aggiornato. 0 in caso di successo, -1 altrimenti (si ha un fallimento se per esempio non c'è spazio sufficiente per ospitare la variabile d'ambiente aggiornata)
OUTPUT	per esempio non c'è spazio sufficiente per ospitare la variabile d'ambiente aggiornata)

FUNZIONE	int unsetenv (char *name)
DESCRIZIONE	Elimina una variabile d'ambiente.
PARAMETRI	• char *name → nome della variabile d'ambiente
OUTPUT	0 in caso di successo, -1 altrimenti (si ha un fallimento se per esempio la variabile era già <u>non</u> presente nell'ambiente)

→ I programmi che lavorano sui thread richiedono che, per essere compilati, sulla command line si scriva `gcc funzione.c -lpthread`

FUNZIONE	<code>int pthread_create(pthread_t *tid, pthread_attr_t *attr, void *(*funct)(void *), void *arg )</code>
DESCRIZIONE	Crea un nuovo thread.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>pthread_t *tid</code> → puntatore all'area di memoria che dovrà ospitare il codice di identificazione del nuovo thread (<code>pthread_t</code> è un <code>unsigned int</code>).</li> <li>• <code>pthread_attr_t *attr</code> → puntatore a una tabella che contiene delle informazioni di startup del nuovo thread.</li> <li>• <code>void *(*funct)(void *)</code> → puntatore alla funzione del codice (che restituisce un <code>void *</code> e prende in input un <code>void *</code>) da cui il thread deve iniziare l'esecuzione.</li> <li>• <code>void *arg</code> → puntatore generico da passare come parametro di input al nuovo thread tramite un registro di CPU.</li> </ul>
OUTPUT	0 in caso di successo; un valore differente altrimenti

FUNZIONE	<code>void pthread_exit(void *status)</code>
DESCRIZIONE	Richiede la terminazione del thread chiamante.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>void *status</code> → puntatore all'area di memoria che contiene le informazioni di terminazione</li> </ul>
OUTPUT	/

FUNZIONE	<code>int pthread_join(pthread_t tid, void **status)</code>
DESCRIZIONE	Attende la terminazione di uno specifico thread.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>pthread_t tid</code> → identificatore del thread di cui si attende la terminazione.</li> <li>• <code>void **status</code> → puntatore al puntatore al buffer contenente il codice di uscita.</li> </ul>
OUTPUT	Identificatore del thread terminato in caso di successo, -1 altrimenti.

FUNZIONE	<code>pthread_t pthread_self(void)</code>
DESCRIZIONE	Ritorna l'identificatore del thread chiamante.
PARAMETRI	/
OUTPUT	Identificatore del thread in caso di successo, -1 altrimenti

FUNZIONE	<code>int pthread_detach(pthread_t thread)</code>
DESCRIZIONE	Contrassegna uno specifico thread come "distaccato": d'ora in poi non ci saranno altre tracce interessate a sincronizzarsi con lui. Un effetto è che, una volta che questo thread termina, le sue informazioni riguardo la terminazione vengono eliminate.
PARAMETRI	<code>• pthread_t thread</code>
OUTPUT	0 in caso di successo, un valore differente altrimenti

## CPU - SCHEDULING

SYSTEM CALL	int nice (int incr)
DESCRIZIONE	Effettua un incremento o decremento della niceness del processo chiamante.
PARAMETRI	<ul style="list-style-type: none"><li>• int incr → intero compreso tra -20 e 19; se è al di fuori di questo intervallo, viene impostato al valore più vicino nel range.</li></ul>
OUTPUT	

SYSTEM CALL	int setpriority (int which, int who, int prio)
DESCRIZIONE	Reimposta la niceness di uno o più processi.
PARAMETRI	<ul style="list-style-type: none"><li>• int which → valore che indica se si opera su un unico processo (PRIO_PROCESS), su un gruppo di processi (PRIO_GROUP) oppure sui processi attivi per conto di uno specifico utente (PRIO_USER).</li><li>• int who → valore che indica il target (e.g. PID)</li><li>• int prio → niceness da attribuire al target.</li></ul>
OUTPUT	

SYSTEM CALL	int getpriority (int which, int who)
DESCRIZIONE	Richiede la niceness di uno o più processi.
PARAMETRI	<ul style="list-style-type: none"><li>• int which</li><li>• int who</li></ul>
OUTPUT	Niceness del target in caso di successo

SYSTEM CALL	<code>int sched_setscheduler(pid_t pid, int policy, const struct sched_param *p)</code>
DESCRIZIONE	Reimposta la priorità di uno specifico thread.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>pid_t pid</code></li> <li>• <code>int policy</code> → classe di priorità da assegnare al thread target, che può essere Time-Sharing (SCHED_OTHER) oppure Real Time.</li> <li>• <code>const struct sched_param *p</code> → puntatore alla tabella in cui viene stabilito qual è la reale gestione dello scheduling del thread target da parte del Kernel. Una delle informazioni che si possono trovare in questa struct è la nice ness del thread target.</li> </ul>
OUTPUT	

SYSTEM CALL	<code>int sched_getscheduler(pid_t pid)</code>
DESCRIZIONE	Richiede la priorità di uno specifico thread.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>pid_t pid</code></li> </ul>

OUTPUT Priorità del thread target in caso di successo

SYSTEM CALL	<code>int sched_setaffinity(pid_t pid, size_t cpusetsize, const cpu_set_t *mask)</code>
DESCRIZIONE	Imposta l'affinità di uno specifico thread nella regola di scheduling.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>pid_t pid</code></li> <li>• <code>size_t cpusetsize</code> → dimensione della maschera di bit che indica le CPU permesse per il thread di cui viene passato il pid.</li> <li>• <code>const cpu_set_t *mask</code> → puntatore alla maschera di bit</li> </ul>
OUTPUT	

SYSTEM CALL	<code>int sched_getaffinity(pid_t pid, size_t cpuset_size, cpu_set_t *mask)</code>
DESCRIZIONE	Chiede al Kernel la maschera di bit che indica le CPU permesse per uno specifico thread.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>pid_t pid</code></li> <li>• <code>size_t cpuset_size</code></li> <li>• <code>cpu_set_t *mask</code></li> </ul>
OUTPUT	Maschera di bit che indica le CPU permesse per il thread di cui viene passato il pid

## VIRTUAL FILE SYSTEM

SYSTEM CALL	<code>int setuid(uid_t id)</code>
DESCRIZIONE	Cambia l'utente corrente dell'applicazione che sta girando. È un comando eseguibile solo dall'utente root (identificato col codice numerico 0) a meno che il programma ha come proprietario il root e ha il bit relativo a SUID impostato. <sup>1</sup>
PARAMETRI	• <code>uid_t id</code> → codice numerico del nuovo utente per conto di cui si vuole far girare l'applicazione.
OUTPUT	

SYSTEM CALL	<code>uid_t getuid()</code>
DESCRIZIONE	Chiede al Kernel qual è l'utente corrente dell'applicazione che sta girando.
PARAMETRI	/
OUTPUT	Codice numerico dell'utente per conto di cui l'applicazione sta girando.

SYSTEM CALL	<code>int creat(char *file_name, int mode)</code>
DESCRIZIONE	Invoca la creazione di un file con conseguente apertura di un canale di I/O verso questo file.
PARAMETRI	• <code>char *file_name</code> → puntatore alla stringa che indica il nome del file. • <code>int mode</code> → intero che specifica i permessi di accesso in codifica ottale (e.g. 0666 corrisponde a 110 110 110). • Descrittore del file in caso di successo, -1 altrimenti (si ha un fallimento se per esempio il vettore degli i-node è saturo).
OUTPUT	

SYSTEM CALL	<code>int open (char *file_name, int option_flags [, int mode])</code>
DESCRIZIONE	Apri un file mantenendo in setup una sessione.
	<ul style="list-style-type: none"> <li>• <code>char *file_name</code></li> <li>• <code>int option_flags</code> → modalità con cui si vuole lavorare sul file che si sta aprendo. Alcuni esempi sono:           <ul style="list-style-type: none"> <li>→ <code>O_RDONLY</code>: sola lettura</li> <li>→ <code>O_WRONLY</code>: sola scrittura</li> <li>→ <code>O_RDWR</code>: lettura e scrittura</li> <li>→ <code>O_APPEND</code>: ogni operazione sarà effettuata solo alla fine del file</li> <li>→ <code>O_CREAT</code>: l'apertura può avvenire anche se il file non è già esistente (in tal caso avviene una creazione)</li> <li>→ <code>O_TRUNC</code>: se il file è già esistente, il suo contenuto viene eliminato</li> <li>→ <code>O_EXCL</code>: flag di esclusività; se combinato con <code>O_CREAT</code>, indica che la creazione del file deve avvenire in modo esclusivo, per cui, se esisteva già, la system call fallisce. La combinazione di questi flag rappresenta un modo per verificare l'esistenza progressiva di quello specifico file.</li> </ul> </li> </ul>
PARAMETRI	<ul style="list-style-type: none"> <li>Questi flag possono essere combinati tramite l'operatore "  ".</li> <li>• <code>int mode</code> [FACOLTATIVO] → intero che specifica i permessi di accesso al file in caso di creazione contestuale all'apertura.</li> </ul>
OUTPUT	Codice numerico del canale che permette di accedere al file in caso di successo, -1 altrimenti

SYSTEM CALL	<code>int close(int descriptor)</code>
DESCRIZIONE	Chiude un canale di I/O. La sessione corrispondente viene chiusa solo nel caso in cui non è accessibile da altri canali validi.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>int descriptor</code> → codice numerico del canale che si sta per chiudere</li> </ul>
OUTPUT	-1 in caso di fallimento
SYSTEM CALL	<code>ssize_t read(int descriptor, char *buffer, size_t size)</code>
DESCRIZIONE	Legge un file tramite uno specifico canale di I/O.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>int descriptor</code></li> <li>• <code>char *buffer</code> → puntatore al buffer dove memorizzare i byte letti</li> <li>• <code>size_t size</code> → dimensione del buffer</li> </ul>
OUTPUT	Numero di byte effettivamente letti in caso di successo, -1 altrimenti (si ha un fallimento se per esempio viene passato come parametro il descrittore di un canale non valido)
SYSTEM CALL	<code>ssize_t write(int descriptor, char *buffer, size_t size)</code>
DESCRIZIONE	<p>Scrive su un file.</p> <p>NB: In particolar modo se il file pointer non punta alla fine del file, questa funzione non aggiunge dei nuovi byte, bensì li sovrascrive.</p>
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>int descriptor</code></li> <li>• <code>char *buffer</code> → puntatore al buffer da cui prendere i byte da scrivere sul file</li> <li>• <code>size_t size</code> → quantità di byte da scrivere</li> </ul>
OUTPUT	Numero di byte effettivamente prelevati dal buffer ed emessi verso il file in caso di successo, -1 altrimenti

SYSTEM CALL	<code>off_t lseek(int descriptor, off_t offset, int option)</code>
DESCRIZIONE	Riposiziona il file pointer.
PARAMETRI	<ul style="list-style-type: none"> <li>• int descriptor</li> <li>• off_t offset → numero di caratteri di cui si vuole spostare il file pointer a partire dal punto indicato dal parametro option (è positivo se lo spostamento è in avanti, negativo altrimenti).</li> <li>• int option → intero che indica l'opzione di spostamento: dall'inizio (0), dalla posizione corrente (1) o dalla fine (2).</li> </ul>
OUTPUT	Nuovo valore del file pointer in caso di successo, -1 altrimenti

SYSTEM CALL	<code>int dup(int descriptor)</code>
DESCRIZIONE	Duplica un canale instantaneamente una copia all'interno della tabella dei descrittori che punterà alla stessa sessione. La copia viene sempre inserita nella prima posizione libera della tabella dei descrittori. In particolare, se viene posizionata nel canale 0, che è quello utilizzato dalla <code>scanf()</code> nel caso in cui il buffer di input sia vuoto, la <code>scanf</code> va a leggere il contenuto del file accessibile dal canale 0 e, quindi, dal canale passato in input nella funzione <code>dup()</code> (RIDIREZIONE DI CANALE).
PARAMETRI	<ul style="list-style-type: none"> <li>• int descriptor</li> </ul>
OUTPUT	Nuovo descrittore in caso di successo, -1 altrimenti

SYSTEM CALL	<code>int dup2 (int oldfd, int newfd)</code>
DESCRIZIONE	Duplica un canale instanzianone una copia in una entry a scelta della tabella dei descrittori.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>int oldfd</code> → descrittore del canale che si vuole duplicare</li> <li>• <code>int newfd</code> → posizione nella tabella dei descrittori in cui il canale deve essere duplicato</li> </ul>
OUTPUT	-1 in caso di fallimento

SYSTEM CALL	<code>int fcntl (int fd, int cmd, ...)</code>
DESCRIZIONE	Controlla quale farà essere la reale operatività da attuare sulla sessione associata al file descriptor (e quindi al canale) fornito come parametro.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>int fd</code> → file descriptor su cui si vuole operare</li> <li>• <code>int cmd</code> → comando da eseguire per la gestione della sessione. Per esempio, è possibile controllare la possibilità di aprire o meno sessioni concorrenti verso lo stesso file a partire da processi concorrenti attraverso il comando LOCKING (specificato col flag F_SETLK/W): questa operazione, se ci sono più sessioni relative a un canale, ne blocca qualcuna a vantaggio delle altre.</li> <li>• Eventuali altri parametri che dipendono dal comando cmd</li> </ul>
OUTPUT	

SYSTEM CALL	<code>int link(const char *oldpath, const char *newpath)</code>
DESCRIZIONE	Crea un nuovo hard link per uno specifico file.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>const char *oldpath</code> → nome già esistente del file (può anche contenere la specifica del percorso di appartenenza, ovvero il path)</li> <li>• <code>const char *newpath</code> → nuovo nome (può anche contenere la specifica del percorso di appartenenza)</li> </ul>
OUTPUT	-1 in caso di fallimento

SYSTEM CALL	<code>int unlink(const char *pathname)</code>
DESCRIZIONE	Rimuove un hard link di uno specifico file. Anche se si tratta dell'unico hard link, dopo questa operazione il file esiste ancora perché il suo i-node è ancora raggiungibile dalla sessione corrente.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>const char *pathname</code> → nome del file (con eventuale specifica del percorso di appartenenza) associato all'hard link che si vuole rimuovere</li> </ul>
OUTPUT	-1 in caso di fallimento

SYSTEM CALL	<code>int symlink(const char *oldpath, const char *newpath)</code>
DESCRIZIONE	Crea un nuovo soft link per uno specifico file.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>const char *oldpath</code></li> <li>• <code>const char *newpath</code></li> </ul>
OUTPUT	-1 in caso di fallimento

SYSTEM CALL	<code>int mkdir(const char *pathname, mode_t mode)</code>
DESCRIZIONE	Creare una directory.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>const char *pathname</code> → puntatore a carattere che indica il nome della directory da creare ed eventualmente da quali altre directory essa dipende.</li> <li>• <code>mode_t mode</code> → le tre triplette che indicano i permessi d'accesso</li> </ul>
OUTPUT	

SYSTEM CALL	<code>int rmdir (const char *pathname)</code>
DESCRIZIONE	Rimuovere una directory.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>const char *pathname</code></li> </ul>
OUTPUT	

SYSTEM CALL	<code>int chmod (const char *path, mode_t mode)</code>
DESCRIZIONE	Manipola i permessi di accesso di un file che non deve essere necessariamente aperto.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>const char *path</code></li> <li>• <code>mode_t mode</code></li> </ul>
OUTPUT	

SYSTEM CALL	<code>int fchmod(int filedes, mode_t mode)</code>
DESCRIZIONE	Manipola i permessi di accesso di un file che deve essere aperto.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>int filedes</code> → descrittore del file di cui si vuole cambiare i permessi.</li> <li>• <code>mode_t mode</code></li> </ul>
OUTPUT	

FUNZIONE	int fsync(int fd)
DESCRIZIONE	Richiede che i dati in buffer cache (ma non i metadati) di uno specifico file vengano riportati sul dispositivo di memoria di massa e porta il thread chiamante nello stato di blocco finché questa operazione non verrà completata.
PARAMETRI	• int fd
OUTPUT	

FUNZIONE	int fdatasync(int fd)
DESCRIZIONE	Richiede che sia i dati in buffer cache di uno specifico file, sia i suoi metadati nella cache degli i-node vengano riportati sul dispositivo di memoria di massa e porta il thread chiamante nello stato di blocco finché questa operazione non verrà completata.
PARAMETRI	• int fd
OUTPUT	

## COMUNICAZIONE TRA THREAD/PROCESSI

SYSTEM CALL	int pipe(int fd[2])
DESCRIZIONE	Crea una PIPE.
PARAMETRI	<ul style="list-style-type: none"><li>int fd[2] → puntatore a un buffer di due interi, che sono: → fd[0]: descrittore di lettura dalla PIPE, che consente di acquisire dati tramite la funzione read()</li><li>→ fd[1]: descrittore di scrittura sulla PIPE, che consente di immettere dati tramite la funzione write()</li></ul>
OUTPUT	-1 in caso di fallimento

FUNZIONE	int mKfifo(char *name, int mode)
DESCRIZIONE	Crea una named PIPE (= FIFO).
PARAMETRI	<ul style="list-style-type: none"><li>char *name → nome della FIFO da creare (tipicamente è un percorso nome).</li><li>int mode → intero che specifica le modalità di creazione e i permessi di accesso alla FIFO.</li></ul>
OUTPUT	L'unico descrittore per avviare alla named PIPE in caso di successo, -1 altrimenti

SYSTEM CALL	<code>int msgget (Key_t Key, int flag)</code>
DESCRIZIONE	<p>Creare / aprire una coda di messaggi.</p> <ul style="list-style-type: none"> <li>• <code>Key_t Key</code> → codice numerico per identificare in modo univoco la coda di messaggi</li> <li>• <code>int flag</code> → flag che indica i permessi di accesso e la modalità di creazione, la quale può essere:</li> </ul>
PARAMETRI	<ul style="list-style-type: none"> <li>→ <code>IPC_CREAT</code>: indica che si sta tentando di creare una coda con la chiave "Key" ma, se esiste già, ci si accontenta di aprirla. <small>Se Key è già avviato a posiziون di una nuova coda che sarà privata (accessibile solo al chiamante e ai suoi eventuali figli)</small></li> <li>→ <code>IPC_EXCL</code>: indica che si vuole esclusivamente creare una coda con la chiave "Key"; se esiste già, la system call fallisce.</li> </ul>
OUTPUT	Descrittore di coda in caso di successo, -1 altrimenti

SYSTEM CALL	<code>int msgctl (int ds_coda, int cmd, struct msqid_ds *buff)</code>
DESCRIZIONE	Esegue un comando cmd su una specifica coda di messaggi.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>int ds_coda</code> → descrittore della coda su cui si vuole operare</li> <li>• <code>int cmd</code> → comando da eseguire. Ci sono tre possibilità: <ul style="list-style-type: none"> <li>→ <code>IPC_RMID</code>: rimozione della coda</li> <li>→ <code>IPC_STAT</code>: richiesta di informazioni riguardanti lo stato della coda</li> <li>→ <code>IPC_SET</code>: impostazione di informazioni riguardanti la gestione della coda (e.g. i permessi di accesso)</li> </ul> </li> <li>• <code>struct msqid_ds *buff</code> → puntatore a una struttura contenente le informazioni riguardanti la coda di messaggi</li> </ul>
OUTPUT	-1 in caso di fallimento

SYSTEM CALL	int msgsnd(int ds_coda, const void *buff, size_t nbyte, int flag)
DESCRIZIONE	Spedisce un messaggio verso una specifica coda.
PARAMETRI	<ul style="list-style-type: none"> <li>• int ds_coda</li> <li>• const void *buff → puntatore al buffer che contiene il messaggio</li> <li>• size_t nbyte → taglia del messaggio</li> <li>• int flag → opzione di spedizione, che può essere bloccante (0) o non bloccante (IPC_NOWAIT).</li> </ul>
OUTPUT	-1 in caso di fallimento

SYSTEM CALL	int msgrcv(int ds_coda, void *buff, size_t nbyte, long type, int flag)
DESCRIZIONE	Estrae un messaggio da una specifica coda.
PARAMETRI	<ul style="list-style-type: none"> <li>• int ds_coda</li> <li>• void *buff</li> <li>• size_t nbyte → numero massimo di byte che si vuole ricevere. Se la taglia del messaggio letto M è maggiore, la parte in più viene buttata, perché M deve comunque essere estratto per intero dalla coda.</li> <li>• long type → valore intero che può rientrare in uno dei seguenti tre casi: <ul style="list-style-type: none"> <li>→ type &gt; 0: indica il tipo di messaggio da ricevere.</li> <li>→ type = 0: indica che deve essere estratto il messaggio più vecchio, seguendo la politica First In First Out.</li> <li>→ type &lt; 0: indica l'insieme dei tipi di messaggio che possono essere considerati per la ricezione.</li> </ul> </li> </ul>
OUTPUT	-1 in caso di fallimento

## GESTIONE DELLA MEMORIA

SYSTEM CALL	<pre>void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset)</pre>
DESCRIZIONE	Mappa delle pagine contigue all'interno dell'address space del processo chiamante. <ul style="list-style-type: none"><li>• void *addr → punto del contenitore di memoria in cui inizia la regione da mappare; se non corrisponde all'inizio di una pagina, verrà effettuato un arrotondamento per difetto.</li><li>• size_t length → quantità di memoria da mappare; se non corrisponde a un multiplo delle taglie delle pagine (= 4096 byte sui processori x86), verrà effettuato un arrotondamento per eccesso.</li><li>• int prot → operazioni che potranno essere eseguite sulle pagine da mappare (PROT_EXEC, PROT_READ, PROT_WRITE, PROT_NONE).</li><li>• int flags → opzioni sulla mappatura da effettuare. Alcuni esempi sono:<ul style="list-style-type: none"><li>→ MAP_PRIVATE: le pagine saranno ereditabili tramite fork() ma saranno protette dallo schema copy on write.</li><li>→ MAP_SHARED: le pagine saranno ereditabili tramite fork() e non saranno protette dallo schema copy on write.</li><li>→ MAP_ANONYMOUS: le pagine saranno inizialmente impostate con tutti i bit a zero.</li></ul></li><li>• int fd → descrittore di un eventuale file da allocare nelle pagine.</li><li>• off_t offset → posizione del file pointer da cui iniziare la mappatura dell'eventuale file.</li></ul>
PARAMETRI	Iniziatore del contenitore di memoria in cui effettivamente inizia la prima pagina mappata in caso di successo, NULL altrimenti.
OUTPUT	

SYSTEM CALL	int munmap(void *addr, size_t length)
DESCRIZIONE	De-mappa delle pagine contigue all'interno dell'address space del processo chiamante.
PARAMETRI	<ul style="list-style-type: none"> <li>• void *addr</li> <li>• size_t length</li> </ul>
OUTPUT	-1 in caso di fallimento

SYSTEM CALL	int mprotect(void *addr, size_t length, int prot)
DESCRIZIONE	Modifica i permessi di accesso di alcune pagine.
PARAMETRI	<ul style="list-style-type: none"> <li>• void *addr</li> <li>• size_t length</li> <li>• int prot</li> </ul>
OUTPUT	-1 in caso di fallimento

SYSTEM CALL	int brk(void *addr)
DESCRIZIONE	<p>Sposta il program-break nell'address space del processo chiamante.</p> <p>Una variante è void *sbrk(int ptr_t increment), che accetta come parametro il numero di byte di cui il BRK deve spostarsi e restituisce in output l'indirizzo del BRK aggiornato.</p>
PARAMETRI	<ul style="list-style-type: none"> <li>• void *addr → nuovo indirizzo del BRK</li> </ul>
OUTPUT	-1 in caso di fallimento

SYSTEM CALL	int madvise(void *addr, size_t length, int advice)
DESCRIZIONE	Dà delle specifiche indicazioni sull'uso della memoria.
PARAMETRI	<ul style="list-style-type: none"> <li>• void *addr</li> <li>• size_t length</li> <li>• int advice → flag che specifica le indicazioni sull'uso del = la memoria. Un valore che può assumere è MADV_REMOVE, che dematerializza le pagine precise dagli altri due par= = metri dalla RAM.</li> </ul>
OUTPUT	

SYSTEM CALL	int shmget(Key_t Key, int size, int flag)
DESCRIZIONE	Crea un'area di memoria condivisibile da più processi. <small>→ anche non relazionati</small>
PARAMETRI	<ul style="list-style-type: none"> <li>• Key_t Key → chiave che identifica la memoria condivisibile</li> <li>• int size → taglia della memoria condivisibile</li> <li>• int flag → modalità di creazione (IPC_CREAT, IPC_EXCL) e permessi di accesso</li> </ul>
OUTPUT	Descrittore della memoria condivisa in caso di successo, -1 altrimenti <small>→ è un'entry della tabella IPECT come per le ceste di messaggi</small>

SYSTEM CALL	<code>int shmidl(int ds_shm, int cmd, struct shmid_ds *buff)</code>
DESCRIZIONE	Esegue un comando cmd su una memoria condivisa. • <code>int ds_shm</code> → descrittore della memoria condivisa su cui si vuole operare
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>int cmd</code> → comando da eseguire. Ci sono tre possibilità:           <ul style="list-style-type: none"> <li>→ IPC_RMID: rimozione della memoria condivisa</li> <li>→ IPC_STAT: richiesta di informazioni riguardanti lo stato della memoria condivisa; tra queste figura il numero di processi che condividono i frame fisici di cui è stato passato il descrittore come parametro.</li> <li>→ IPC_SET: impostazione di informazioni riguardanti la gestione della memoria condivisa (e.g. i permessi di accesso)</li> </ul> </li> <li>• <code>struct shmid_ds *buff</code> → puntatore a una struttura contenente le informazioni riguardanti la coda di messaggi</li> </ul>
OUTPUT	-1 in caso di fallimento

SYSTEM CALL	<code>void *shmat(int ds_shm, void *addr, int flag)</code>
DESCRIZIONE	Aggiunge alcuni frame di una memoria condivisa a determinate pagine logiche del processo chiamante; perciò, permette di utilizzare effettivamente questi frame in lettura o in scrittura.
PARAMETRI	<ul style="list-style-type: none"> <li>• <code>int ds_shm</code></li> <li>• <code>void *addr</code> → indirizzo dell'address space da cui si vuole fare partire il collegamento con i frame della memoria condivisa.</li> <li>• <code>int flag</code> → modalità di accesso (SHM_R, SHM_W, SHM_RW)</li> </ul>
OUTPUT	Indirizzo effettivo dell'address space del chiamante per l'accesso alla memoria condivisa in caso di successo, -1 altrimenti

SYSTEM CALL	int schedet(const void *addr)
DESCRIZIONE	Sgancia alcuni frame di una memoria contirisa da determinate pagine logiche del processo chiamante; queste pagine non saranno più valide.
PARAMETRI	• const void *addr
OUTPUT	-1 in caso di fallimento

N.B.: è necessario includere la libreria pthread

## SINCRONIZZAZIONE

PER GESTIRE LA SERVATURA:  
spin-lock (& unlock)  
spin-unlock (& lock)

FUNZIONE	int pthread_spin_init(pthread_spinlock_t *lock, int pshared)
DESCRIZIONE	Inizializza una variabile di tipo spinlock_t, che è una struttura contenente una locazione per la serratura di una certa sezione critica e altri metadati che, per esempio, tengono traccia di qual è stato l'ultimo thread a impostare effettivamente la serratura a 1.
PARAMETRI	<ul style="list-style-type: none"><li>• pthread_spinlock_t *lock → puntatore alla variabile da inizializzare</li><li>• int pshared → intero che indica il modo con cui la nuova variabile viene inizializzata. Ci sono due possibilità:<ul style="list-style-type: none"><li>→ PTHREAD_PROCESS_SHARED: la variabile verrà usata per sincronizzare thread di processi differenti.</li><li>→ PTHREAD_PROCESS_PRIVATE: la variabile verrà usata per sincronizzare thread della stessa applicazione.</li></ul></li></ul>
OUTPUT	

SYSTEM CALL	int semget(Key_t Key, int size, int flag)
DESCRIZIONE	Crea un array semaforico.
PARAMETRI	<ul style="list-style-type: none"><li>• Key_t Key → chiave che identifica l'array semaforico (IPC_PRIVATE è un caso speciale in cui il vettore sarà anonimo e raggiungibile esclusivamente tramite descrittore).</li><li>• int size → numero di entry dell'array semaforico</li><li>• int flag → modalità di creazione (IPC_CREAT, IPC_EXCL) e permessi di accesso</li></ul>
OUTPUT	Descrittore dell'array semaforico in caso di successo, -1 altrimenti

SYSTEM CALL	<code>int semctl(int fd-sem, int sem-num, int cmd, union semun arg)</code>
DESCRIZIONE	Esegue un comando cmd su un array semaforico. <ul style="list-style-type: none"> <li>• int fd-sem → descrittore dell'array semaforico su cui si vuole operare</li> <li>• int sem-num → indice dell'elemento dell'array su cui si vuole operare</li> <li>• int cmd → comando da eseguire. Ci sono sette possibilità:<ul style="list-style-type: none"> <li>→ IPC_RMID: rimozione dell'array semaforico</li> <li>→ IPC_STAT: richiesta di informazioni riguardanti lo stato del vettore semaforico <small>metadati</small></li> <li>→ IPC_SET: impostazione di informazioni riguardanti la gestione dell'array semaforico (e.g. i permessi di accesso)</li> </ul></li> <li>• union semun arg → puntatore al buffer con eventuali parametri per il comando cmd, che possono essere:<ul style="list-style-type: none"> <li>→ int val se cmd=SETVAL</li> <li>→ struct semid_ds *buff se cmd=IPC_STAT o cmd=IPC_SET</li> <li>→ ushort *array se cmd=GETALL o cmd=SETALL</li> </ul></li> </ul>
PARAMETRI	
OUTPUT	-1 in caso di fallimento

SYSTEM CALL	<code>int semop(int ds-sem, struct sembuf oper[], int number)</code>
DESCRIZIONE	<p>Effettua un'operazione di sincronizzazione su un array semaforico.</p> <ul style="list-style-type: none"> <li>• <code>int ds-sem</code></li> <li>• <code>struct sembuf oper[]</code> → array di strutture dati; ciascuna sua entry corrisponde a un'operazione su un particolare distributore dell'array semaforico, e viene definita così:</li> </ul> <pre>struct sembuf {     ushort sem_num;     short sem_op;     short sem_flg; };</pre> <p>→ <code>ushort sem_num</code>: indice dell'array semaforico su cui effettuare l'operazione</p> <p>→ <code>short sem_op</code>: valore intero che può rientrare in uno dei seguenti tre casi:</p> <ul style="list-style-type: none"> <li>▷ <code>sem_op &gt; 0</code>: indica il numero di gettoni che il chiamante sta per rilasciare.</li> <li>▷ <code>sem_op &lt; 0</code>: indica il numero di gettoni che il chiamante sta provando a prelevare.</li> <li>▷ <code>sem_op = 0</code>: indica che il chiamante verrà messo in attesa finché il valore di quello specifico distributore non sarà esattamente uguale a 0 (sincronizzazione sullo 0).</li> </ul> <p>→ <code>short sem_flg</code>: modalità con cui l'operazione deve essere eseguita. Alcuni esempi sono:</p> <ul style="list-style-type: none"> <li>▷ <code>O</code>: l'operazione è bloccante se non ci sono gettoni nel momento in cui si tenta un prelevo o se ce n'è almeno uno quando si tenta una sincronizzazione sullo 0.</li> <li>▷ <code>IPC_NOWAIT</code>: l'operazione non è mai bloccante.</li> <li>▷ <code>SEM_UNDO</code>: quando il chiamante termina, le operazioni che aveva eseguito sull'array semaforico vengono revocate dal kernel.</li> </ul> <ul style="list-style-type: none"> <li>• <code>int number</code> → numero di entry valide nell'array <code>oper[]</code></li> </ul>
OUTPUT	-1 in caso di fallimento

FUNZIONE	<code>int sem_init(sem_t *sem, int pshared, unsigned int value)</code>
DESCRIZIONE	Inizializza un semaforo (named o unnamed).
PARAMETRI	<ul style="list-style-type: none"> <li><code>sem_t *sem</code> → puntatore alla struttura dati semaforica di livello user</li> <li><code>int pshared</code></li> <li><code>unsigned int value</code> → valore con cui il nuovo semaforo viene inizializzato</li> </ul>
OUTPUT	

FUNZIONE	<code>sem_t *sem_open(const char *name, int flag)</code>
DESCRIZIONE	Definisce un semaforo named.
PARAMETRI	<ul style="list-style-type: none"> <li><code>const char *name</code> → percorso nome del semaforo</li> <li><code>int flag</code> → permessi di accesso</li> </ul>
OUTPUT	Puntatore alla struttura dati semaforica di livello user in caso di successo

FUNZIONE	<code>int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr)</code>
DESCRIZIONE	Inizializza un mutex.
PARAMETRI	<ul style="list-style-type: none"> <li><code>pthread_mutex_t *mutex</code> → puntatore al mutex</li> <li><code>const pthread_mutexattr_t *attr</code> → puntatore a una tabella di attributi del mutex</li> </ul>
OUTPUT	

## EVENTI

SYSTEM CALL	int Kill (int pid, int segnale)
DESCRIZIONE	Invia un segnale a un altro processo/thread. • int pid → identificatore del processo/thread che deve ricevere il segnale
PARAMETRI	• int segnale → codice numerico del segnale da inviare -1 in caso di fallimento che, per esempio, si può avere nel
OUTPUT	caso in cui il chiamante gira per conto di un utente che non ha i permessi per inviare segnalazioni a chiunque.

FUNZIONE	int raise (int segnale)
DESCRIZIONE	Invia un segnale al thread chiamante.
PARAMETRI	• int segnale
OUTPUT	0 in caso di successo

FUNZIONE	unsigned alarm (unsigned time)
DESCRIZIONE	Invia il segnale SIGALRM al main thread del processo chiamante; se il main thread è già terminato, viene colpita un'altra traccia della medesima applicazione.
PARAMETRI	• unsigned time → tempo allo scadere del quale il segnale SIGALRM viene inviato
OUTPUT	tempo restante prima che arrivi un eventuale segnale SIGALRM invocato precedentemente

→ In effetti, il Kernel può passare ad massimo una variabile intera all'interno della fotografia di cui come parameter di gestore di l'evento, nel momento in cui quest'ultimo viene a essere attivato.

SYSTEM CALL	<code>void *signal (int sig, void (*ptr)(int)) (int)</code>
DESCRIZIONE	Cambia lo stato di relazione tra il chiamante e una specifica segnalazione. LINES: È l'intero processo
PARAMETRI	<ul style="list-style-type: none"> <li>• int sig</li> <li>• void (*ptr)(int) → parametro che può assumere tre valori:           <ol style="list-style-type: none"> <li>1) CASO IN CUI LA SEGNALAZIONE DEVE ESSERE CATTURATA: puntatore alla funzione di gestione del segnale che può accettare un intero come unico parametro* e il cui valore di ritorno non deve essere consegnato a nessuno.</li> <li>2) CASO IN CUI LA SEGNALAZIONE DEVE ESSERE IGNORATA IMPLICATAMENTE: SIG_DFL</li> <li>3) CASO IN CUI LA SEGNALAZIONE DEVE ESSERE IGNORATA ESPlicitAMENTE: SIG_IGN</li> </ol> </li> </ul>
OUTPUT	Puntatore alla funzione di gestione del segnale che era in vigore prima della chiamata alla system call se il processo si trovava nello stato di cattura; SIG_DFL se prima la segnalazione era ignorata implicitamente; SIG_IGN se prima la segnalazione era ignorata esplicitamente; SIG_ERR in caso di fallimento.

SYSTEM CALL	<code>int pause (void)</code>
DESCRIZIONE	Mantiene il thread chiamante nello stato di blocco finché non sopraggiunge un segnale qualunque.
PARAMETRI	
OUTPUT	Sempre -1



SYSTEM CALL	<code>int sigprocmask(int how, const sigset_t *set, sigset_t *oset)</code>
DESCRIZIONE	Imposta la gestione della signal mask. • int how → comando che indica il modo con cui il Kernel dovrà gestire la signal mask. Può assumere uno dei seguenti tre valori: → SIG_BLOCK: aggiunge degli elementi alla lista dei segnali da bloccare. → SIG_UNBLOCK: rimuove degli elementi dalla lista dei segnali da bloccare. → SIG_SETMASK: ridefinisce gli elementi della lista di segnali da bloccare.
PARAMETRI	• const sigset_t *set → puntatore alla signal mask che consente di selezionare i segnali su cui operare. • sigset_t *oset → puntatore all'area di memoria in cui verrà scritto il valore della signal mask prima delle modifiche.
OUTPUT	-1 in caso di fallimento

FUNZIONE	<code>int sigemptyset(sigset_t *set)</code>
DESCRIZIONE	Imposta a zero tutti i bit della signal mask.
PARAMETRI	• sigset_t *set
OUTPUT	

FUNZIONE	<code>int sigfillset(sigset_t *set)</code>
DESCRIZIONE	Imposta a uno tutti i bit della signal mask.
PARAMETRI	• sigset_t *set
OUTPUT	



FUNZIONE	int sigaddset(sigset_t *set, int signo)
DESCRIZIONE	Imposta a 1 uno specifico bit della signal mask.
PARAMETRI	<ul style="list-style-type: none"><li>• sigset_t *set</li><li>• int signo</li></ul>
OUTPUT	

FUNZIONE	int sigdelset(sigset_t *set, int signo)
DESCRIZIONE	Imposta a 0 uno specifico bit della signal mask.
PARAMETRI	<ul style="list-style-type: none"><li>• sigset_t *set</li><li>• int signo</li></ul>
OUTPUT	

FUNZIONE	int sigismember(sigset_t *set, int signo)
DESCRIZIONE	Controlla il valore di uno specifico bit della signal mask.
PARAMETRI	<ul style="list-style-type: none"><li>• sigset_t *set</li><li>• int signo</li></ul>
OUTPUT	

SYSTEM CALL	int sigpending(sigset_t *set)
DESCRIZIONE	Restituisce l'insieme dei segnali che sono pendenti.
PARAMETRI	<ul style="list-style-type: none"><li>• sigset_t *set</li></ul>
OUTPUT	-1 in caso di fallimento





SYSTEM CALL	int sigaction(int sig, const struct sigaction *act, struct sigaction *oact)
DESCRIZIONE	Gestisce lo stato di relazione tra il processo chiamante e una specifica segnalazione sig, ed eventualmente blocca altri segnali mentre sig viene processato tramite un gestore.
PARAMETRI	<ul style="list-style-type: none"><li>• int sig</li><li>• const struct sigaction *act → puntatore a una struttura che include il nuovo stato di relazione e le informazioni su come deve essere gestita la signal mask mentre sig viene processato tramite un handler.</li><li>• struct sigaction *oact → puntatore a una struttura dove verranno memorizzate le impostazioni precedenti.</li></ul>
OUTPUT	-1 in caso di fallimento

## SOCKET

SYSTEM CALL	int socket(int domain, int type, int protocol)
DESCRIZIONE	Crea un socket.
PARAMETRI	<ul style="list-style-type: none"> <li>• int domain → dominio del nuovo socket</li> <li>• int type → tipo di comunicazione del nuovo socket</li> <li>• int protocol → protocollo del nuovo socket ; se impostato a 0, viene selezionato il protocollo di default.</li> </ul>
OUTPUT	<p>Descrittore del nuovo socket in caso di successo, -1 altrimenti      (si ha un fallimento se per esempio la tabella dei descrittori è già      saturata o se il protocollo selezionato per la coppia dominio-tipo di      comunicazione non è disponibile).</p>

SYSTEM CALL	int bind(int ds_sock, struct sockaddr *my_addr, int addrlen)
DESCRIZIONE	Assegna un identificatore (indirizzo) a uno specifico socket.
PARAMETRI	<ul style="list-style-type: none"> <li>• int ds_sock → descrittore del socket</li> <li>• struct sockaddr *my_addr → puntatore alla struttura dati che definisce l'indirizzo da assegnare al socket.</li> <li>• int addrlen → lunghezza in byte dell'indirizzo da assegnare al socket.</li> </ul>
OUTPUT	-1 in caso di fallimento (che, per esempio, si verifica se viene passato un descrittore di socket non valido o se un indirizzo già impegnato per un altro socket).



SYSTEM CALL	int accept (int ds_sock, struct sockaddr *addr, int *addrlen)
DESCRIZIONE	Accetta una connessione che entrerà in uno specifico socket A. È un servizio bloccante e, affinché non fallisca, A deve trovarsi nello stato di LISTENING.
PARAMETRI	<ul style="list-style-type: none"><li>• int ds_sock → descrittore del socket A</li><li>• struct sockaddr *addr → puntatore alla struttura in cui verrà memorizzato l'indirizzo del socket B che si conterrà.</li><li>• int *addrlen → puntatore all'area di memoria in cui verrà scritta la taglia dell'indirizzo di B.</li></ul>
OUTPUT	Descrittore del nuovo socket generato dinamicamente in caso di successo, -1 altrimenti.

SYSTEM CALL	int listen (int ds_sock, int backlog)
DESCRIZIONE	Mette uno specifico socket nello stato di listening.
PARAMETRI	<ul style="list-style-type: none"><li>• int ds_sock</li><li>• int backlog → numero di connessioni che sarà possibile mantenere in sospeso; è un valore solo orientativo, poiché il sistema operativo potrebbe decidere di consentire anche un numero più alto di connessioni pendenti.</li></ul>
OUTPUT	-1 in caso di fallimento



SYSTEM CALL	int connect(int ds_sock, struct sockaddr *addr, int addrlen)
DESCRIZIONE	<ul style="list-style-type: none"><li>• Se il tipo di comunicazione è STREAM, porta uno specifico socket B a richiedere la connessione con un altro socket A</li></ul>
PARAMETRI	<ul style="list-style-type: none"><li>• Se il tipo di comunicazione è_DGRAM, imposta il socket A come destinatario fisso dei datagrammi predetti da uno specifico socket B.</li><li>• int ds_sock → descrittore del socket B</li><li>• struct sockaddr *addr → puntatore alla struttura contenente l'indirizzo del socket A</li><li>• int addrlen → taglia dell'indirizzo di A</li></ul>
OUTPUT	-1 in caso di fallimento

SYSTEM CALL	int send(int sock.ds, const void *buff, int size, int flag)
DESCRIZIONE	Invia dati in output tramite socket. È una variante della system call write() e necessita di una precedente chiamata a connect().
PARAMETRI	<ul style="list-style-type: none"><li>• int sock.ds → descrittore del socket che invia i dati.</li><li>• const void *buff → puntatore al buffer da cui prendere i byte da spedire.</li><li>• int size → quantità di byte da spedire</li><li>• int flag → opzioni di spedizione; una possibilità è che, in caso di errore severo, non venga emesso alcun segnale (che avrebbe potuto essere SIGPIPE, il quale indica che si sta inviando dati in un tubo di connessione senza che però ci sia alcun socket pronto ad acquisire informazioni).</li></ul>
OUTPUT	-1 in caso di fallimento





SYSTEM CALL	int recv(int sock_des, void *buff, int size, int flag)
DESCRIZIONE	Acquisisce dati in input tramite socket. È una variante della system call read().
PARAMETRI	<ul style="list-style-type: none"><li>• int sock_des → descrittore del socket che acquisisce i dati.</li><li>• void *buff → puntatore al buffer dove memorizzare i byte letti.</li><li>• int size → dimensione del buffer</li><li>• int flag → ESEMPIO: MSG_WAITALL, per cui il chiamante non esca dello stato di blocco finché nel buffer non saranno disponibili tutti i size byte.</li></ul>
OUTPUT	-1 in caso di fallimento

SYSTEM CALL	int sendto(int sock_des, const void *buff, int size, int flag, struct sockaddr *addr, int addrlen)
DESCRIZIONE	Invia dati in output verso uno specifico socket. Non necessita di una precedente chiamata a connect() e può essere invocata solo per le comunicazioni DGRAM.
PARAMETRI	<ul style="list-style-type: none"><li>• int sock_des</li><li>• const void *buff</li><li>• int size</li><li>• int flag</li><li>• struct sockaddr *addr → puntatore alla struttura contenente l'indirizzo del socket destinazione</li><li>• int addrlen → taglia dell'indirizzo del socket destinazione</li></ul>
OUTPUT	-1 in caso di fallimento



SYSTEM CALL	int recvfrom(int sock_fd, void *buff, int size, int flag, struct sockaddr *addr, int *addrlen)
DESCRIZIONE	Acquisisce dati in input da uno specifico socket. Può essere invocata solo per le comunicazioni DGRAM.
PARAMETRI	<ul style="list-style-type: none"><li>• int sock_fd</li><li>• void *buff</li><li>• int size</li><li>• int flag</li><li>• struct sockaddr *addr</li><li>• int *addrlen</li></ul>
OUTPUT	-1 in caso di fallimento

FUNZIONE	int socketpair(int domain, int type, int protocol, int sockvec[2])
DESCRIZIONE	Crea una coppia di socket già connessi tra loro. Può essere invocata solo per il dominio AF_UNIX.
PARAMETRI	<ul style="list-style-type: none"><li>• int domain</li><li>• int type</li><li>• int protocol</li><li>• int sockvec[2] → vettore in cui verranno memorizzati i descrittori dei due nuovi socket.</li></ul>
OUTPUT	-1 in caso di fallimento

FUNZIONE	struct hostent *gethostbyname (const char *name)
DESCRIZIONE	Richiede informazioni riguardanti l'host con uno specifico nome tra cui i suoi IP number. Il servizio encapsulato in questa API è definito DNS (Domain Name Service) ed è distribuito: initialmente richiede informazioni sull'host a un processo P all'interno del sistema; se P non lo conosce, lo chiede a sua volta a un altro processo P' all'interno della rete, e così via.
PARAMETRI	• const char *name → nome dell'host di cui vengono richieste le informazioni
OUTPUT	Puntatore alla struttura dati in cui verranno memorizzate le informazioni riguardanti l'host (in particolare, gli IP number saranno reperibili nell'array di puntatori **h_addr_list all'interno della struct).



SYSTEM CALL	<code>int setsockopt(int sockfd, int level, int optname, void *optval, socklen_t optlen)</code>
DESCRIZIONE	Reimposta una specifica opzione di uno o più socket. • <code>int sockfd</code> → descritto del socket su cui si vuole operare. • <code>int level</code> → intero che indica se l'opzione deve essere aggiornata solo per il socket <code>sockfd</code> oppure anche per quelli con lo stesso protocollo. • <code>int optname</code> → tipo di opzione da reimpostare. Alcuni esempi sono: → <code>SO_RCVTIMEO</code> : opzione per impostare il timeout per le op- erazioni di lettura tramite socket (che di fatto sono bloccanti). → <code>SO_REUSEADDR</code> : opzione che, se attivata, fa sì che, appena il socket viene chiuso, il suo indirizzo venga recuperato im- mediatamente per essere assegnato a un altro socket (= operazione che altrimenti viene eseguita periodicamente e non in modo sincrono rispetto alla istruzione dell'oggetto di I/O).
PARAMETRI	<code>void *optval</code> → puntatore al buffer in cui viene specificato come deve essere impostata l'opzione. <code>socklen_t optlen</code> → dimensione del buffer <code>optval</code>
OUTPUT	0 in caso di successo, -1 altrimenti.

SYSTEM CALL	<code>int getsockopt(int sockfd, int level, int optname, const void *optval, socklen_t *optlen)</code>
DESCRIZIONE	Chiede lo stato di una specifica opzione di uno o più socket. • <code>int sockfd</code> • <code>int level</code>
PARAMETRI	<code>int optname</code> • <code>const void *optval</code> • <code>socklen_t optlen</code>
OUTPUT	0 in caso di successo, -1 altrimenti.



Vedere anche le API correlate nell'ultimissima slide e sulla pagina man.

FUNZIONE	int inet_aton(const char *cp, struct in_addr *inp)
DESCRIZIONE	Converte un indirizzo IP nella forma "x.y.z.t" in un effettivo codice numerico.
PARAMETRI	<ul style="list-style-type: none"><li>const char *cp → stringa nella forma "x.y.z.t" che esprime un particolare indirizzo IP.</li><li>struct in_addr *inp → puntatore alla struttura in cui verrà memorizzato il codice numerico relativo all'indirizzo IP.</li></ul>
OUTPUT	