INTERFACCIA (API) DI GESTIONE DEI PROCESSI

- **FUNZIONE FORK** – BIFORCAZIONE DEL PROCESSO IN PADRE E FIGLIO

pid_t fork ()

/* Biforca il processo in padre e figlio: al figlio restituisce sempre 0; al padre restituisce il pid > 0 del figlio biforcato; la funzione restituisce –1 se la biforcazione del processo fallisce (restituisce solo al padre, ovviamente, visto che il figlio non è stato biforcato), per esempio per insufficienza di memoria per la creazione del processo figlio, o altro motivo */

- **PROCEDURA EXIT** – TERMINAZIONE DEL PROCESSO

void exit (int stato)

/* Termina il processo, con stato di terminazione pari a "stato"; di solito si esegue "exit(0)" per terminare il processo con stato di uscita indicante correttezza, "exit(–1)" per terminare indicando errore; la scelta dei significati degli stati di uscita è tuttavia arbitraria e lasciata al programmatore; la procedura "exit" non fallisce mai, fa sempre terminare il processo che la invoca, qualunque sia il valore di "stato". Parametri: **stato** è un intero indicante lo stato di uscita del processo; solo gli 8 bit meno significativi contano */

- **FUNZIONE EXECV** – MUTAZIONE DI CODICE ESEGUIBILE DEL PREOCESSO

int execv (char * file, char * argv[])

/* Muta il codice di un processo: il processo assume il codice eseguibile del programma risiedente nel file eseguibile "file"; la funzione restituisce 0 se il codice è stato mutato; la funzione restituisce –1 se c'è stato un errore, cioè se la mutazione di codice non è stata eseguita. Parametri: **file** è il puntatore alla stringa contenente il nome del file; **argv** è un vettore di puntatori a stringa: ogni stringa rappresenta un parametro da passare; per convenzione **argv[0]** dovrebbe puntare al nome del programma stesso, cioè alla stringa "file; i puntatori agli argomenti effettivi vanno da **argv[1]** ad **argv[n]** (per un totale di $n \geq 0$ argomenti); e infine si deve avere **argv[n+1] = NULL**, per marcare la fine */

- **FUNZIONE EXECL** – MUTAZIONE DI CODICE ESEGUIBILE DEL PREOCESSO

int execl (char * file, char * arg0, char * arg1, char * arg2, ..., char * argn+1)

/* Come "execv", ma i puntatori a stringa **argx** (con $0 \leq x \leq n + 1$) si passano uno per uno; la funzione "execl" ha pertanto un numero variabile di argomenti */

- **FUNZIONE WAIT** – ATTESA DELLA TERMINAZIONE DI UN FIGLIO QUALSIASI

pid_t wait (int * stato)

/* Mette un processo padre in stato di attesa dell'evento di terminazione di un processo figlio qualsiasi, e restituisce il pid del figlio effettivamente terminato; la funzione restituisce –1 se il processo padre esce dalla "wait" per un errore che non abbia a che vedere con la terminazione di un figlio; l'argomento "stato" assume il valore di "exit" del processo figlio terminato. Parametri: **stato** è il puntatore a un intero, che conterrà lo stato di uscita del figlio terminato (contano solo gli 8 bit meno significativi; gli 8 bit più significativi vengono impostati da parte del sistema operativo stesso) */

- **FUNZIONE WAITPID** – ATTESA DELLA TERMINAZIONE DEL FIGLIO "PID"

pid_t waitpid (pid_t pid, int * stato, int opzioni)

/* Mette un processo padre in stato di attesa dell'evento di terminazione del processo figlio avente pid pari a "pid", e ne restituisce il pid; la funzione è insensibile alla terminazione di eventuali altri figli aventi pid diverso da "pid"; la funzione restituisce –1 se il processo padre esce dalla "waitpid" per un errore che non abbia a che vedere con la terminazione del figlio "pid"; l'argomento "stato" assume il valore di "exit" del processo figlio terminato. Parametri: **pid** è il "pid" del figlio della cui terminazione il padre si mette in attesa; **stato** è il puntatore a un intero, che conterrà lo stato di uscita del figlio terminato (solo gli 8 bit meno significativi); **opzioni** è un intero che specializza la funzione "waitpid", e di solito vale 0 (altrimenti vedere il manuale in linea della funzione, perché le opzioni possibili sono parecchie) */

Come biforcare un processo:

```
if (fork ()) == 0) { /* qui avviene la biforcazione! */
     /* istruzioni eseguite da parte del processo figlio */
     exit (...); /* il processo figlio termina con stato di uscita ... */
} else {
     /* istruzioni eseguite da parte del processo padre */
} /* if */
```

Come biforcare un processo e mettere il padre in attesa della terminazione del figlio:

```
if (fork ()) == 0) { /* qui avviene la biforcazione! */
     /* istruzioni eseguite da parte del processo figlio */
     exit (...); /* il processo figlio termina con stato di uscita ... */
} else {
     wait (&stato); /* il padre attende la terminazione del figlio */
     /* istruzioni eseguite da parte del processo padre */
} /* if */
```

Come biforcare un processo e affidare al figlio il compito di eseguire un programma:

```
if (fork ()) == 0) { /* qui avviene la biforcazione! */
     execl (file, ...); /* il processo figlio muta codice eseguibile */
     exit (-1); /* se si arriva qui, vuol dire che la exec è fallita! */
} else {
     /* istruzioni eseguite da parte del processo padre */
} /* if */
```

Come biforcare un processo e affidare al figlio il compito di eseguire un programma, passando al programma un elenco di n ³ 0 argomenti, usando la funzione execv:

```
if (fork ()) == 0) { /* qui avviene la biforcazione! */
     char * argv[MAX_ARG_NUM];
     /* file, 1° arg, ..., n^esimo arg sono tutti puntatori a stringa */
     argv[0] = file; /* convenzionale, ma di solito si fa così */
     argv[1] = 1° argomento ... (una stringa)
     argv[2] = 2° argomento ... (una stringa)
     ...
     argv[n-1] = (n-1)^esimo argomento ... (una stringa)
     argv[n] = n^esimo argomento ... (una stringa)
     argv[n+1] = NULL; /* necessario, serve a terminare l'elenco! */
     execv (file, argv);
     exit (-1); /* se si arriva qui, vuol dire che la exec è fallita! */
} else {
     /* istruzioni eseguite da parte del processo padre */
} /* if */
```

Come biforcare un processo e affidare al figlio il compito di eseguire un programma, passando al programma un elenco di n ³ 0 argomenti, usando la funzione execl:

```
if (fork ()) == 0) { /* qui avviene la biforcazione! */
     /* file, 1° arg, ..., n^esimo arg sono tutti puntatori a stringa */
     execl (file, file, 1° arg, 2° arg, ..., (n-1)^esimo arg, n^esimo arg, NULL);
     exit (-1); /* se si arriva qui, vuol dire che la exec è fallita! */
} else {
     /* istruzioni eseguite da parte del processo padre */
} /* if */
```

NOTA BENE: quando il programma "file" partirà, verrà attivata la funzione `main (int argc, char * argv[])` contenuta nel programma stesso, il vettore `argv` verrà automaticamente dimensionato come `argv[n + 1]`, l'argomento `argc` varrà precisamente `n + 1`, e gli elementi dell'argomento `argv` avranno i valori seguenti: `argv[0] == file`; `argv[1] == 1° arg`, ..., `argv[n] = n^esimo arg`; ATTENZIONE: l'argomento `argv[n + 1]` NON RISULTA DEFINITO nell'ambiente della funzione `main`!