En aquesta segona entrega hem incluït un fitxer jocs.c i un jocs.h que contenen el codi i els prototipus de les funcions. S'han hagut de fer les modificacions pertinents al Makefile per tal que aquests formessin part de user.c.

Les funcions que proporcionem per provar el còdi són:

```
void provar_fork (int num_fills);
void provar_get_stats (void);
void provar_exit(void);
void provar_nice (void);
void provar_switch (void);
void provar_semafors(void);
void run2_jp(void);
```

I s'haura de descomentar del main() de user.c la que volguem provar.
A continuació una descripció de cada funció.

void provar_fork (int num_fills):
A n'aquesta funció li passarem un num_fills que és el nombre de fills que volem crear a partir del procés pare. Els crea consecutivament, és a dir, tots seràn fills del procés 0. Per cada creació és treuen per pantalla dues frases per cada procés creat i finalment el total de fills creats:

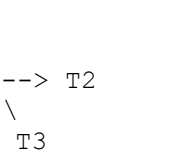
```
Soc el fill, tinc PID: 1
Return del fork: 1 -> Soc el pare, tinc PID:0
...
Total de fills creats: 1
```

void provar_get_stats (void):
Comprova que la crida al sistema get_stats funcioni correctament.
- Provem amb num de pid negatiu. Ha de donar error.
- Provem amb num de procés que no existeix. Ha de donar error.
- Provem amb el pid del qui crida, i una @ de resultat invàlida: Error.
- Provem amb una @ de resultat fora del nostre espai: Error
- Provem a crear un fill amb fork() i mirar el seu stats.
- Provem de mirar el nostre temps de cpu.

void provar_exit(void):
Comprova la crida al sistema exit().
- Fem un fork i provem d'assassinar al pare. No s'ha de poder.
- Assassinem al fill.

void provar_nice (void):
Prova d'executar les crides a nice() amb valors positius, negatius i 0.
Aquesta prova es pot extendre dins provar_switch. Ho comentem a continuació.

void provar_switch (void):
Aquesta funció crea un arbre de 4 processos on cada procés imprimeix Tpid on PID és el num de pid del procés, 0, 1, 2, 3. No tots són fills de T0.



Es pot observar com tots van canviant escrivint el diferent valor. A més aquesta prova permet molt més. Es pot afegir codi entre els tags:
/*Codi de T2 */
/*Fi Codi T2 */

Com per exemple modificar els valors del quantum de cada procés amb el nice, i es comprovarà com van canviant de contexte i estant més temps o menys a la cpu.

A més també es pot experimentar amb els semàfors, colocant esperes, destruccions i altres al codi.

void provar_semafors(void):
Prova la implementació i comportament dels semàfors.
- Provarem 1 a 1 les crides del sistema sem_xx amb diferents valors per cobrir tots els casos possibles d'errors i creació i destrucció, suma i resta d'un semàfor.
- Un cop provades les funcions bàsiques crearem un fill i farem que aquest s'esperi fent que el pare hagi inicialitzat el semàfor a 0. El pare farà un for(); per esperar uns moments, intentarà fer un sem_destroy però no podrà perquè el fill encara està a la cua. Finalment farà un sem_signal, alliberant al fill.

void run2_jp(void):
Són les mateixes proves que vam entregar a l'entrega E1. L'explicació erà la següent (mateixa que E1 també):

/**ENTERGA E1**/

El joc de proves que hem fet s'ha d'activar descomentant la linia run2_jp() que hi ha al main del fitxer user.c.
Llavors s'ha de descomentar la funcionalitat que es vulgui provar:

/*EXCEPCIONS*/
GENERAL PROTECTION
Prova l'excepció de protecció general, és la que sortia al codi user.c original.

DIVIDE ERROR
Intenta dividir per 0 i genera una excepcio de divisio per 0.

PAGE FAULT
Accedeix directament a pila movent-hi un valor aleatori i provoca un Page Fault

GENERACIO DE INTERRUPCIONS DES DE CODI D'USUARI
Acces a una interrupcio de sistema desde usuari (p.ex floating point error)
Substituir el nombre de interrupcio per provar-ne d'altres

Acces a una int de usuari (p.ex divide error)

```
/*CRIDES A SISTEMA*/
   Es pot descomentar el perror() del final per veure com queda l'ERRNO.
  Escriu la frase
  write(1,"Prova d'un write",256);

  No escriu res
  write(1,"",100);

  Bad FD
  write(-10,"Hola",10);

  Bad size
  k = write(1,"Hola",-10);
  Imprimeix el num de caracters uqe li posem a baix. Ara esta amb 10, per
imprimir-los
  tots s'ha de posar un 683, tambe podem provar p.ex amb 1000, encara que a la
M.V.
  avegades falla. Despres diu quants se n'han escrit.
  /* int k = write(1,"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\
n"
        "bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb\n"
        "ccccccccccccccccccccccccccccccccccccccccccccccccccccccccc\n"
        "ddddddddddddddddDDDDDdddDDDdddDDDddddddDDDDDddDDDdddDDDd\n"
        "eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee\n"
        "fffffffffffffffffffffffffffffffffffffffffffffffffffffffff\n"
        "gggggggggggggggggggggggggggggggggggggggggggggggggggggggggg\n"
        "hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh\n"
        "iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii\n"
        "jjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj\n"
        "kkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkk\n"
        "En aquest bloc de text hi ha 683 caracters",10);
   printf("\n");
   printint(k);
  */

  Bad ADDRESS
  write(1,k,200);

  Descomentar les funcions per veure com esta errno o per imprimir l'enter k
  perror();
  printint(k);
  PROVA DE LA FUNCIO PRINTF i PRINTI
        Aquestes funcions no s'havien de fer, i per aixo no assegurem encara
        el funcionament 100%. Hem detectat algun bug a printint.
    char *text1 = "p";
    int k = printf(text1);
    printf("\nMida bytes escrits: ");
    printint(k);
```

Felip Moll
Josep Martí