Examen de rattrapage

L'objectif est de lancer un processus dont la fonction est la mise en mémoire d'une valeur entière. Lorsqu'un signal est émis cette valeur stockée est affichée. L'implémentation proposée est constituée de deux processus dont le fonctionnement est ainsi schématisé, le premier correspond à l'exécutable `memoire.exe`, le deuxième à l'exécutable `cache.exe`. Les deux processus se déroulent dans la même fenêtre.

texte entré par utilisateur	texte affiché	Premier processus (P1)	Deuxième processus (P2)
`memoire ecrire 20`		P1 est lancé	
	20 mis en mémoire	P1 réalise l'affichage	
		P1 créé un signal	
		nommé `SIGNAL_FERMETURE`	
		avec le statut non-signalé	
		P1 lance P2 avec 20 en argument	
			P2 lancé reçoit 20 en argument
entrée		fin du processus P1	
`memoire lire`		P1 est à nouveau lancé	
		le signal `SIGNAL_FERMETURE`	
		a le statut signalé	
			P2 réceptionne l'activation du signal `SIGNAL_FERMETURE`
	valeur=20		affichage de la valeur par P2
			fin du processus P2
entrée		fin du processus P1	

Exercice 1 *Écrivez deux programmes* `memoire1.c` *et* `cache1.c` *tel que le processus associé à* `memoire1.exe` *lance un processus associé à* `cache1.exe`*. Le deux processus font de l'affichage et s'arrêtent tout seul.*

Exercice 2 *Écrivez deux programmes* `memoire2.c` *et* `cache2.c` *tel que la commande* `memoire2 ecrire 20` *lance un processus associé à* `cache2.exe` *et transmet l'entier 20. Le deux processus font de l'affichage et s'arrêtent tout seul.*

Exercice 3 *Écrivez deux programmes* `memoire3.c` *et* `cache3.c` *tel que la commande* `memoire3 ecrire 20` *lance un processus associé à* `cache3.exe` *et transmet l'entier 20. Ce processus créé un signal nommé* `SIGNAL_FERMETURE`*. Le deuxième processus arrive à ouvrir ce signal créé. Le deux processus font de l'affichage et s'arrêtent tout seul.*

Exercice 4 *Écrivez deux programmes* `memoire4.c` *et* `cache4.c` *de façon à réaliser complètement l'objectif.*

Solution pour le programme principal

```
#include <string.h>
#include <stdio.h>
#include <Windows.h>
VOID  test_erreur(LPCTSTR msg_etape);
INT main(INT argc, CHAR * argv[]) {
  STARTUPINFO start_info;
  PROCESS_INFORMATION process_info;
  INT valeur=0;
  HANDLE hSignal;
  CHAR ligne[100];
  memset(&start_info, 0, sizeof start_info);
  start_info.cb = sizeof start_info;
  if (1 == argc) {
    printf("les arguments possibles sont : ");
    printf("ecrire suivi d'un entier ou lire\n");
    getchar();  return 0;
  }
  else if (3 == argc) {
    if (0 == strcmp(argv[1], "ecrire")) {
```

```
      valeur=atoi(argv[2]);
      printf("mise en memoire de la valeur %i\n",valeur);
      if (NULL==CreateEvent(NULL, FALSE, FALSE, "SIGNAL_FERMETURE"))
        test_erreur("CreateEvent");
      sprintf_s(ligne,sizeof(ligne),"cache.exe %i",valeur);
      if (!CreateProcess(NULL,ligne, NULL, NULL, FALSE, 0, NULL, NULL,\
        &start_info, &process_info))
        test_erreur("CreateProcess");
      if (!CloseHandle(process_info.hProcess)) test_erreur("CloseHandle process");
      if (!CloseHandle(process_info.hThread)) test_erreur("CloseHandle thread");
    }
    else {
      printf("mot %s inconnu\n",argv[1]);
      getchar(); return 0;
    }
  }
  else if (2==argc) {
    if (0 == strcmp(argv[1], "lire")) {
      hSignal=OpenEvent(EVENT_ALL_ACCESS,FALSE,"SIGNAL_FERMETURE");
      if (NULL==hSignal) test_erreur("OpenEvent");
      if (!SetEvent(hSignal)) test_erreur("SetEvent");
      getchar(); return 0;
    }
    else {
      printf("mot %s inconnu\n",argv[1]);
      getchar(); return 0;
    }
  }
  else {
    printf("nombre d'arguments inconnus");
  }
  getchar(); return 0;
}
```

solution pour le programme exécuté

```
#include <Windows.h>
#include <stdio.h>
#include <stdlib.h>
VOID  test_erreur(LPCTSTR msg_etape);
INT main(INT argc,char *argv[]) {
  DWORD dwWait;
  HANDLE signal_fermeture;
  INT valeur;
  if (2!=argc) {
    printf("erreur de fonctionnement le programme aurait dû recevoir un argument\n");
  }
  valeur=atoi(argv[1]);
  printf("mise en memoire effectue de la valeur %i\n",valeur);
  signal_fermeture = OpenEvent(EVENT_ALL_ACCESS,FALSE, "SIGNAL_FERMETURE");
  if (NULL == signal_fermeture) test_erreur("OpenEvent");
  dwWait = WaitForSingleObject(signal_fermeture, INFINITE);
  if (WAIT_OBJECT_0 == dwWait) test_erreur("WaitForSingleObject");
  printf("valeur en memoire est %i\n",valeur);
  return 0;
```

```
}
```

A Annexe

Voici le coeur du code à compléter.

```
#include <string.h>
#include <stdio.h>
#include <Windows.h>
VOID  test_erreur(LPCTSTR msg_etape);
INT main(INT argc, CHAR * argv[]) {
  /*******************************************************A COMPLETER */
  if (1 == argc) {
    printf("les arguments possibles sont : ");
    printf("ecrire suivi d'un entier ou lire\n");
  }
  else if (3 == argc) {
    if (0 == strcmp(argv[1], "ecrire")) {
      valeur=atoi(argv[2]);
      printf("mise en memoire de la valeur %i\n",valeur);
      /*************************************************A COMPLETER */
    }
    else {
      printf("mot %s inconnu\n",argv[1]);
    }
  }
  else if (2==argc) {
    if (0 == strcmp(argv[1], "lire")) {
      /*************************************************A COMPLETER */
    }
    else {
      printf("mot %s inconnu\n",argv[1]);
    }
  }
  else {
    printf("nombre d'arguments inconnus");
  }
  getchar(); return 0;
}
```

voici le code de `test_erreur.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <Windows.h>
VOID  test_erreur(LPCTSTR msg_etape) {
  DWORD  erreur;
  LPTSTR  msg_erreur;
  INT    code_severity, code_facility, code_erreur;
  PCHAR  severity[4] = { "Success", "Informational", "Warning", "Error" };
  PCHAR  facility[11] = { "FACILITY_NULL",
    "FACILITY_RPC",    "FACILITY_DISPATCH",    "FACILITY_STORAGE","FACILITY_ITF",
    "",    "",    "FACILITY_WIN32", "FACILITY_WINDOWS","","FACILITY_CONTROL" };
  PCHAR  msg_facility;
  if ((erreur = GetLastError()) == NO_ERROR)  return;
```

```
  code_severity = (0xC0000000 & erreur) >> 30;
  code_facility = (0x0FFF0000 & erreur) >> 16;
  code_erreur = 0x0000FFFF & erreur;
  if (code_facility <= 10)
    msg_facility = facility[code_facility];
  else
    msg_facility = "FACILITY_???";
  FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
    FORMAT_MESSAGE_FROM_SYSTEM,
    NULL, erreur,
    MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
    (LPTSTR)&msg_erreur, 0, NULL);
  printf("\n*** ERREUR ***     Code erreur : %8.8lX\n\n"
    "  Etape : %s\n  Erreur : %s"
    "  (Severity : %s   Facility : %s   Error : %d)\n\n",
    erreur, msg_etape, msg_erreur,
    severity[code_severity], facility[code_facility], code_erreur);
  LocalFree(msg_erreur);
  ExitProcess(code_severity);
}
```