empêche les autres de prendre la main. Mais ça ne les bloque pas. Ils sont ralentis. De façon systématique ce processus reprendra la main et épuisera un quantum de temps à chaque reprise.

Pour l'arrêter, une interruption est nécessaire : par exemple `Ctrl C` ou `Ctrl \` dans un système multi-tâche (cf. Unix) ou `kill -9 1234` ou `1234` est le pid du processus infernal.

Solution 4 1. il est le seul processus utilisateur,

2. tous les autres sont en attente d'entrée-sortie,
3. il est le plus prioritaire.

Pour l'enchaînement des opérations, voir le cours.

Solution 5 *Les réponses ne sont pas détaillées. dans chaque cas, il faut se demander ce qui se passerait si tout utilisateur pouvait exécuter à sa guise les instructions ou codes correspondants. Attention quand même : pour ce qui concerne la date du jour ou l'allocation mémoire, ce ne sont pas des instructions simples, mais des programmes ou des fonctions.*

masquer une interruption	*oui*	lire la date du jour	*non*
modifier la date du jour	*oui*	modifier l'allocation mémoire	*oui*
appeler l'ordonnanceur	*oui*	appel d'une fonction en C	*non*

Solution 6 1.
```
afficher "Nb de paramètres (arguments) : " argc " n"
Pour i=0   argc-1
  afficher i ":" arg [i] " n"
afficher " n ariables d'en ironnement : n"
i=0
 q arge[i] != NULL
  afficher i ":" arge[i] " n"
  i++
afficher "Nb de  ariables d'en ironnement :" i " n"
```

2.
```
#include <stdio.h>

int main(int argc, char** arg , char** arge){
  printf("Nb de paramètres (arguments) : %d n", argc);
  for(int i=0; i<argc; i++){
    printf("%d : %s n",i, arg [i]);
  }
  printf(" n ariables d'en ironnement : n");
  int i=0;
  while(arge[i] != NULL){
    printf("%d : %s n", i, arge[i]);
    i++;
  }
  printf("Nb de  ariables d'en ironnement : %d n",i);
}
```

Solution 7 Toutes les substitutions sont faites par le `shell` avant de lancer l'exécution. Donc on aura successivement :

- 4 paramètres ;
- nombre des fichiers du répertoire, sauf ceux qui commencent par `.` (point) + 1 (`$PA H`).
- nombre des fichiers du répertoire d'accueil qui commencent par `.` et qui possèdent au moins 2 lettres supplémentaires (ni `.` lui-même ni `..`)

Solution 8 1.
```
i=0
 ROU E=faux
 q arge[i] != NULL et non  ROU E
   ROU E=chercherDebut("PA H=",arge[i]);
  i++
si  non  ROU E
  afficher "Pas de PA H dans les  ariables d'en  !"
  exit
sinon
   ROU E=faux
  i--
   ROU E=rechercher(getcwd() ou "." , arge[i]+taille("PA H="))
  if  ROU E
    afficher "Ce rép. est exécutable!"
  sinon
    afficher "Ce rép. n'est pas exécutable!"
```

2.
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char** arg , char** arge){
  char * motif="PA H=", *delim=":", *trou e=NULL; // UNIX delim=";"
  int i = 0 ;
  while(arge[i] != NULL && arge[i]!=(trou e=strstr(arge[i],motif)))
    i++;
  if (arge[i]==NULL){
    printf("La chaîne %s n'existe pas ! n",motif);
    return 1;
  } else {
    char * cwd=(char *)malloc(1000);
    getcwd(cwd,1000); // répertoire courant

    char * source=arge[i]+strlen(motif);
    char * tok=strtok(source,delim); // appel initial a ec la source
    while(tok && strcmp(tok,cwd) && strcmp(tok,".")){
      tok=strtok(NULL,delim);
    }
    if (tok)
      printf("Ce rép. %s est exécutable grâce   : %s n",cwd, tok);
    else
      printf("Ce rép. %s n'est pas exécutable ! n",cwd);
  }
}
```

Solution 9
1.
```
#include <stdio.h>
#include <stdlib.h>
int fact(int n){
  if (n<=1)
    return 1;
  else
    return n*fact(n-1);
}
int main(int argc, char** arg , char** arge){
  if(argc!=2){
    printf("Syntaxe incorrecte : %s 8  n", arg [0]);
    return 1;
  } else {
    int n=atoi(arg [1]);
    printf("%d ! = %d n",n,fact(n));
    return 0;
  }
}
```
2.
```
fact(3)
  fact(2)
    fact(1)
    ret 1;
  ret 2*1
ret 3*2=6
```

Solution 10
1. On choisit un tableau à 2 dimensions dynamique où la largeur de chaque ligne est égale à son indice +1.
2. **triangle(n)**

```
tab=malloc(n+1 entiers)
Pour i=0   n
  tab[i]=malloc(i+1 entiers)
  tab[i][0]=1
  tab[i][i]=1
  pour j=1   i-1
    tab[i][j]=tab[i-1][j-1]+tab[i-1][j]
```

3.
```
#include <stdio.h>
#include <stdlib.h>
int** triangle(int n){ /* ret un triangle de pascal */
  int** tab=malloc((n+1)*sizeof(int));
  for(int i=0; i<=n; i++){
    tab[i]=malloc((i+1)*sizeof(int));
    tab[i][0]=1;
    tab[i][i]=1;
    for(int j=1; j<i; j++){
      tab[i][j]=tab[i-1][j-1]+tab[i-1][j];
    }
  }
  return tab;
}
 oid afficher riangle(int** tab, int n){
  for(int i=0;i<=n;i++){
    for(int j=0; j<=i; j++){
      printf("%6d ", tab[i][j]);
    }
    printf(" n");
  }
}
int main(int argc, char** arg , char** arge){
  if(argc!=3){
    printf("Syntaxe incorrecte : %s 8 4 n", arg [0]);
    return 1;
  } else {
    int n=atoi(arg [1]);
    int p=atoi(arg [2]);
    int **tab=triangle(n);
    afficher riangle(tab,n);
    printf("Nombre de combinaisons C(%d,%d) = %d n",n,p,tab[n][p]);
    return 0;
  }
}
```

Solution 11