USTL Master mention Informatique

ASE - Contrôle de TD (durée 1h30 – documents autorisés)

Exercice 1: *Questions*

1.1 Proposer une **macro** FREESTACK(CTX,N) qui vérifie que l'espace *non utilisé* dans la pile du contexte CTX soit au moins de N octets (sinon elle provoquera un arrêt du programme).

1.2 (a) Quelle est la taille maximale d'un fichier si l'on prend des blocs/inodes de 256 octets dans l'implantation vue en TD (on prendra pour hypothèse un volume de capacité infinie). Quelle est alors la taille maximale si :

- (b) un bloc/inode fait désormais 512 octets.
- (c) on ajoute une triple indirection de blocs.

1.3 La suite de Syracuse est définie sur $\mathbb{N}$ par :

$$syr(n) = \begin{cases} n/2 & \text{si n pair} \\ 3 \times n + 1 & \text{sinon} \end{cases}$$

La conjecture de syracuse dit que, $\forall n > 0$, si l'on calcule la valeur $syr(\ldots syr(n)\ldots)$ alors on finira toujours par obtenir la valeur 1. Par exemple, si l'on prend $n = 5$, alors $syr(5) = 16$, $syr^2(5) = syr(16) = 8$ et $syr^3(5), syr^4(5), syr^5(5)$ seront respectivement égales à $4, 2, 1$. On sait que la conjecture est vérifiée pour tout entier de 32 bits.

Proposer une méthode utilisant les instructions int setjmp(jmp_buf env) et int longjmp(jmp_buf env) q afin de calculer le *nombre d'appels* alors effectués pour une valeur n initiale avant d'obtenir la valeur 1.

Exercice 2: *Changement de contexte et ordonnancement*

On rappelle la structure qui sert à mémoriser le contexte d'une tâche.

```
struct ctx_s {
  enum   ctx_state_e ctx_status;
  func_t  *ctx_f;
  void    *ctx_args;
  void* ctx_esp;
  void* ctx_ebp;
  unsigned char * ctx_stack;
  struct ctx_s *     ctx_next;
  struct ctx_s *     ctx_list;
#define CTX_MAGIC 0xCAFEBABE
  unsigned ctx_magic;
};
```

On souhaite réaliser un ordonnanceur *avec priorité* : chaque tâche reçoit désormais à sa création une priorité fixe int priorite . Une fonction d'ordonnancement int fo(int priorite) évalue le nombre de cycles à attribuer au processus avant le prochain switch_to_context . La valeur retournée par cette fonction est un entier positif ou nul si le processus ne doit pas être activé dans les circonstances actuelles. On suppose qu'une fois le nombre de cycles attribué à une tache, il ne pourra être réévalué qu'au prochain tour d'anneau.

2.1 Décrire (en français) les structures, puis les fonctions à modifier.

2.2 Réimplanter les fonctions yield et/ou switch_to_ctx.

On suppose désormais que la fonction $\mathrm{int\ fo(int\ priorite\ ,\ int\ elapsed)}$ prend un paramètre supplémentaire $\mathrm{int\ elapsed}$ qui représente le nombre de cycles successifs écoulés pour lesquels le processus n'a pas eu le CPU. On considérera que les cycles incomplets (dus à des interruptions autres que celles générées par le "timer") sont comptés comme des cycles complets.
2.3 (a) Décrire les modifications à apporter et (b) proposer leur implémentation en C.

Exercice 3: *Un mini système de fichiers "à la Unix"*

On souhaite réaliser un programme qui défragmente un système de fichiers. On suppose qu'un défragmenteur rétablit l'ordre de lecture des blocs de données/de structure sur des secteurs successifs pour un cylindre / des cylindres successifs pour chaque fichier d'un volume.
3.1 Pour quelles raisons la fragmentation pénalise t-elle la vitesse d'accès aux données ? Sur notre volume Unix (sans défragmenteur intégré), donner un stratégie de création/effacement de fichiers qui produirait au moins un fichier fragmenté.

3.2 En vous aidant d'un schéma, expliquer comment ordonner *idéalement*, les blocs de données/de structure d'un fichier afin de réaliser une lecture continue du flux.

On suppose que le volume possède au moins 1 bloc libre et que toute écriture est réalisée que si la précédente a eu lieu
3.3 Comment réaliser une défragmentation **sûre** d'un fichier donné ? Quelle est la suite de commandes à réaliser ? Quelles sont alors les données/structures à déplacer en plus de celles du propre fichier ? En quoi est-ce pénalisant ? Quelle stratégie adopter si l'espace libre est plus important ?