

1 Pile d'un contexte

1.1 Vérification

Proposer une définition de macro-commande `STACK_ASSERT(ctx, m)` qui vérifie que la pile sauvegardée dans le tableau `ctx_stack` du contexte `ctx` ait au moins `m` octets disponibles au dernier changement de contexte.

1.2 Réallocation

Comment, selon vous, «réallouer» un buffer de pile `ctx_stack` dont la taille serait insuffisante? Réalisez un schéma en supposant que la taille du buffer soit doublée : quels changements doivent être réalisées sur la liste chaînée des registres `ebp` sauvegardés lors de chaque appel de fonction ?

2 Gestion des processus

On propose dans cette partie de créer dynamiquement des nouveaux processus, plutôt que d'exécuter uniquement ceux présents dans l'anneau des contextes initial. Un *pid* (`pid_t`) est un entier long non signé associé à un processus dont la valeur est unique.

2.1 Processus et *pid* associé

Proposez une implémentation (modifications/ajouts de structures de données et modifications de fonctions) afin d'associer un *pid* à chaque processus. Vous donnerez également une implémentation de la procédure :

```
pid_t getpid(void); /* pid du processus */
```

2.2 Processus père et pid

En supposant que chaque processus de l'anneau possède un processus père dont il est issu, proposez une implémentation (modifications/ajouts de structures de données et fonctions) afin de mémoriser le *pid* du père. Vous donnerez également une implémentation de la procédure :

```
pid_t getppid(void); /* pid du processus père */
```

2.3 Fonction *fork* et création de nouveaux processus

```
pid_t fork(void);
```

Cette fonction lorsqu'elle est appelée crée un processus fils, copie "exacte" du processus père du point de vue de la pile d'exécution au moment de l'appel. Cela signifie qu'une fois le nouveau processus créé, tout se passe comme si les processus père et fils avaient réalisés tous les deux un appel à *fork*. Ainsi, à leur reprise, les deux processus feront un retour d'appel de fonction *fork*. Seule la valeur retournée par *fork* différera : elle sera nulle dans le cas du processus fils, et sera le *pid* du fils dans le cas du processus père.

Remarques : on supposera que tous les processus ont la même taille de buffer de pile donnée par `CTX_STACK`, et qu'une fonction `getnewpid()` nous permet d'obtenir un *pid* non utilisé

- Quelles sont les structures de données à dupliquer pour créer un processus fils. Quelles sont les modifications à apporter posteriori, à la fois sur la structure `ctx_s` et la pile `ctx_stack`. (pensez en particulier à la "liste chaînée des sauvegardes des registres `ebp`")
- A quel point lors de l'exécution de *fork* la pile du processus fils doit être dupliquée/modifiée ? Comment par exemple donner un résultat de *fork* différent pour le processus fils ? Comment rendre l'exécution du fils possible ?

2.4 Fonction *wait* et attente de processus

```
pid_t wait(void);
```

La fonction suspend l'exécution du processus courant jusqu'à ce qu'un fils se termine. Si le processus appelant ne possède aucun fils, la fonction renvoie `-1`. Si le processus possède au moins un fils zombi, la fonction renvoie le `pid` d'un fils zombi avant de le terminer définitivement et l'enlever des tables du système. Si le processus possède des fils mais aucun fils zombi, le processus est bloqué jusqu'à ce que l'un des fils devienne zombi.

- modifiez la fonction `start_current_ctx` afin que les processus *zombis* (terminés) aient leur contexte mémorisés dans une file dédiée aux processus terminés.
- proposez une implémentation de la fonction `pid_t wait()`. Afin qu'un processus bloqué ne reste pas dans l'anneau des contextes actifs, vous proposerez un structure (file par exemple) qui conserve les processus bloqués en attente de la terminaison d'un fils.
- complétez la fonction `start_current_ctx` afin qu'un processus fils qui se termine puisse rechercher un éventuel père bloqué par un appel à *wait*.

2.5 Bonus pour plus tard ...

Si vous implémentez les fonctions *fork/wait* dans votre TP, montrez dans votre compte rendu votre code, ainsi que des tests : j'en tiendrai compte.