Josep Maria Olivé i Albert Vilardell

Document de Seguiment:

Hem pogut aconseguir implementar totalment el nivell 1, és a dir els threads amb els seus corresponents jocs de proves, encara que falta formalitzar-los, i parcialment el nivell 2, ja que el codi dels semàfors està implementat però no els jocs de proves corresponents, i per tant el seu correcte funcionament no està validat.

Pel que fa a les modificacions respecte la entrega de disseny, principalment hem modificat el TCB dels threads, per poder emmagatzemar informació que durant la implementació ens hem adonat que era rellevant, com una llista notifyAtExit pel thread esperant per fer un join amb un altre thread donat, o un enter errno, ja que cada thread farà crides a sistema independentment dels altres, també una variable per emmagatzemar el resultat de sortida un cop el procés ha acabat i espera a que el pare en reculli el resultat. També hem afegit atributs al TCB com joinable, per marcar si es pot fer join o no amb un thread determinat, o pag_userStack que és una variable que emmagatzema la pàgina física en la que es troba la pila d'usuari del thread determinat.

També, hem afegit la crida a sistema getTid(), que retorna el TID del thread en CPU. L'hem utilitzat pels jocs de proba.

Pel que fa a la resta de codi que ja teníem no em fet més modificacions.

Ens hem trobat amb múltiples problemes durant el desenvolupament, a continuació una llista dels més rellevants:

1. Un error en el vector de tasques protected_tasks, que tenia la primera i ultima posició marcades com a memòria no accessible, va fer que en modificar codi sensible per acomodar els threads a ZeOS, no se'ns deixes passar a mode usuari. Vam aconseguir solucionar el problema eliminant les dos línies de codi que marcaven aquestes tasques com a regions no accessibles.
2. En la crida a sistema pthreads_create, deixàvem a la pila d'usuari una direcció de retorn final modificada que, mitjançant una nova crida a sistema s' encarregava de guardar el resultat del thread, fent-lo accessible per a possibles joins. Però en ser la adreça del wrapper de la syscall de mode usuari no se'ns permetia accedir al símbol des de mode sistema. Per tal de solucionar-ho, ara passem a la crida pthreads_create un argument més amb la adreça de la crida a sistema encarregada de fer aquesta funció que finalitza el thread.

Josep Maria Olivé i Albert Vilardell

A continuació podem trobar una taula amb totes les tasques que havíem de realitzar per tal de tenir la nova implementació:

Llegenda: verd → tasca realitzada, taronja → tasca en procés, vermell → tasca pendent

Tasca	Implementació
Creació estructures threads	
Pthread_create i Pthread_exit	
Planificador multi-threading	
Rutines de servei multi-threading	
Testing 1: Validació canvis al codi original pel multi-threading	
Pthread_join (Nivell 1 acabat)	
Testing 2: funcions Pthread	
Codi de semàfors (nivell 2)	
Testing 3: semàfors	
Càrregues de treball	

Per ser més precisos, afegim una taula on avaluem l'estat dels fragments del codi més importants:

Fragment de codi	Implementació	Jocs de proves
pthreads_create()		
pthreads_join()		
pthreads_exit()		
sys_fork()		
sys_exit()		
sem_init()		
sem_wait()		
sem_post()		
sem_destroy()		
scheduling()		

El scheduling no consta només d'una funció en concret com la resta, sinó que correspon a tot el fitxer sched.c, des de la implementació del scheduling fins la inicialització dels processos idle i init.
sys_fork() i sys_exit() han estat avaluats pel Testing 1, que inclou només el cas 1 thread/procés.
El testing 2 prova les crides a pthreads amb només la creació d'un sol thread. El testing 3 prova els semàfors juntament amb dos threads.
Aquests jocs de proves **es troben dins el fitxer user.c** i es poden provar canviant el valor de la variable **selected**.