OPERATING SYSTEM

ASSIGNMENT 6

SIGNAL HANDLING IN PINTOS

SAYAN MANDAL (14CS30032)
SOURAV PAL (14CS10062)

************************Files Changed**

1) signal.c
2) signal.h
3) thread.c
4) thread.h

***********************Data Structures Added ************************************

Added a global unblock_list to store the threads which have been received SIG_UNBLOCK signal and sigmask is set and SIG_IGN is not defined for this signal.

In thread.h every thread has now following list:

- **Signal_list :** stores pending signals to be delivered to the corresponding thread and
- **Child_list :** stores the children of the corresponding thread.
- Child list_elem to insert the thread into child_list , unblock list_elem to insert the thread into the global unbloock_list ,
- Parentid to store the is of the parent thread , cpu_limit_over bit to denote if the SIG_CPU signal is called or not.
- Lifetime field to denote the lifetime set by the function setlifetime().
- Mylifetime field to store the time ticks spent by the thread from its creation till now.
- **In signal.h every signal has following field list:**
    - **Sent_id :** to denote the id of the sending thread .
    - **Receive_id :** to denote the id of the receiving thread.
    - **Sigtype :** to denote the type of the signal(SIG_CPU,SIG_CHLD etc.)
- **In signal.h every sigset has following field list:**
    - Five signals , we will denote by setting and resetting the corresponding field to denote the sigset.

***************************DESIGN***************************************

- We kept a mask field of 4 bits , which is initialized as 4 bits set . If we call SIG_IGN , then we reset the bit corresponding to the signal .
- **Implementing signal(sig_name,sig_mode) call:**
- In this call we just get the current thread by calling thread_current() and set the mask corresponding to the sig_mode and sig_type . we just ignore the SIG_KILL here.
- **Implementing kill(thread_id,sig_type):**
- For SIG_KILL signal we are first checking if the thread_id is a child of thread_current() by traversing the child_list of thread_current().
- If the sig_type is SIG_CPU we just the set the cpu_limit_over field of the thread which will denote that that SIG_CPU is pending for that thread.
- If the signal is SIG_CHLD , then we just simply push the signal into the signal_list of the thread .
- We just traverse the signal_list of the thread and replace any same signal with the current signal .
- **Implementing sigchild_handler(thread_id):**
- First we check the mask bit for sigchild , if it is 0 then we just ignore, else we just update the child_died field of the thread and print the stats.
- **Implementing sigkill_handler(thread_id):**
- We just call thread_exit()
- **Implementing sigcpu_handler(thread_id):**
- First we check if SIG_IGN is on by checking the mask value of the thread . If it is set then we just ignore , else we call thread_exit().
- **Implementing sigunblock_handler(thread_id):**
- First we check if SIG_IGN is on by checking the mask value of the thread . If it is set then we just ignore , else we check if the thread is block, if it is so then we just call thread_unblock.
- **Implementing sigemptyset(sigset) and sigfillset(sigset):**
- For sigemptyset we just reset every field of the sigset and for sigfillset we just set every field of the sigset.
- **Implementing sigaddset(sigset,signum) and sigdelset(sigset,signum):**
- For sigaddset we just set the corresponding signal to 1 and for sigdelset we just reset the corresponding signal to 0.
- **Implementing sigprocmask(how,sigset,sigset):**
- Implemented as linux.
- **Handling Signals:**
- We are handling every signals after the call of thread_schedule_tail() as the context switch happens and a thread gets the cpu before executing the thread we are first handling all the signals that are pending in the signal_list of the thread and removing those signals from the signal_list.

- And for every signals except SIG_KILL we are checking blockmask field of the thread to check if that signal is blocked or not . If the signal is blocked we are not calling the signal_handler , else we are calling the signal_handler.
- We are traversing the unblock_list and for every thread if that signal is not blocked and sig_ignored then we are calling sigunblock_handler().
- **Update in thread_tick() :**
- We are incrementing the mylife field of the thread by 1 in every thread_tick and if the mylife is > lifetime of the thread , then we are setting the cpu_limit_over field of that thread to 1.

    - We are initializing every field of the thread in the init_thread function and setting the parent_id of the thread ot the id of the thread_current() and increasing the child of thread_current() by 1.
    - We have added a function init_thread_initial() for the initial_thread that will initialize the field of the initial_thread for ex. The parentid for the initial_thread will be the thread_id and lifetime will be set to 0 as in the thread_tick() we are doing if the lifetime of the thread is 0 then we are ignoring SIG_CPU signals.
    - In the thread_exit() , we are first checking if the thread is not initial_thread , in case of initial_thread we are not calling kill(sigchild), else we are calling kill(sigchild) and we are checking for the existence of the parent thread in kill function. We are first calling a function get_thread(id) for the cooresponding id if the thread doesn't exist , in that case it will return NULL and in that case we just return from the kill function.

Note : We have changed the name of the function from signal to _signal to avoid conflict. Also we have changed the SIG_UNBLOCK option in sigprocmask to SIG_UNBLOCK_ again to avoid conflict.

**************THE END**************