علی کرامتی
علی پاکدل
محمد هنرجو

1. علت غیرفعال کردن وقفه چیست؟
Xv6 page 56 & 57

Interrupts can cause concurrency even on a single processor: if interrupts are enabled, kernel code can be stopped at any moment to run an interrupt handler instead. Suppose iderw held the idelock and then got interrupted to run ideintr. Ideintr would try to lock idelock, see it was held, and wait for it to be released. In this situation, idelock will never be released—only iderw can release it, and iderw will not continue running until ideintr returns—so the processor, and eventually the whole system, will deadlock.

To avoid this situation, if a spin-lock is used by an interrupt handler, a processor must never hold that lock with interrupts enabled.

وقفه می تواند باعث ایجاد هم روندی حتی در یک پردازنده شود. اگر وقفه ها فعال باشند کد هسته می تواند در هر مقطعی متوقف شود تا مدیریت وقفه به جای آن انجام شود که می تواند منجر به deadlock در کل سیستم شود.

توابع pushcli()و popcli() به چه منظور استفاده شده و چه تفاوتی با cli و sti دارند؟

xv6 re-enables interrupts when a processor holds no spin-locks; it must do a little book-keeping to cope with nested critical sections. acquire calls pushcli (1667) and release calls popcli (1679) to track the nesting level of locks on the current processor. When that count reaches zero, popcli restores the interrupt enable state that existed at the start of the outermost critical section. The cli and sti functions execute the x86 interrupt disable and enable instructions, respectively. It is important that acquire call pushcli before the xchg that might acquire the lock (1581). If the two were reversed, there would be a few instruction cycles when the lock was held with interrupts enabled, and an unfortunately timed interrupt would deadlock the system. Similarly, it is important that release call popcli only after the xchg that releases the lock (1581).

xv6 وقفه ها را دوباره فعال می کند وقتی که پردازنده قفل چرخشی را نگه نداشته باشد و برای برآمدن از عهده ی critical section های تو در تو باید داده هایی را ذخیره کند acquire تابع pushcli را فرا می خواند و release تابع popcli را فرا می خواند تا فرایند های تو در توی lock در پردازنده حال حاضر را دنبال کند.توابع cli و sti به ترتیب دستورهای فعال و غیرفعال کردن وقفه ی x86 را اجرا می کنند.

2. مختصری راجع به تعامل میان پردازه ها توسط دو تابع مذکور توضیح دهید.

acquiresleep (4622) uses techniques that will be explained in Chapter 5. At a high level, a sleep-lock has a locked field that is protected by a spinlock, and acquiresleep's call to sleep atomically yields the CPU and releases the spin-lock. The result is that other threads can execute while acquiresleep waits. Because sleep-locks leave interrupts enabled, they cannot be used in interrupt handlers. Because acquiresleep may yield the processor, sleep-locks cannot be used inside spin-lock critical sections (though spin-locks can be used inside sleep-lock critical sections).

در توابع دسته دوم یا همان sleep-lock یه محوطه ی lock دارد که با توابع دسته ی اول یا همان spin-lock محافظت شده و فراخوانی acquiresleep پردازنده را تحویل می دهد و spin-lock را رها می کند در نتیجه بقیه ی ریسه ها می توانند اجرا شوند هنگامی که acquiresleep صبر می کند.به دلیل اینکه توابع دسته دوم یا همان sleep-lock وقفه ها را فعال باز می گذارد نمیتوان در مدیریت وقفه ها از آن استفاده کرد

چرا در مثال تولیدکننده/مصرف کننده استفاده از قفلهای چرخشی ممکن نیست.

چون در قفل های چرخشی وقفه ها غیرفعال اند

3. حالات مختلف پردازه ها در xv6 را توضیح دهید.

UNUSED پردازه ای که تاکنون اجرا نشده و فاقد pid است.

EMBRYO پردازه ای که پس از فراخوانی allocproc دارای pid مخصوص خود شده و به عنوان استفاده شده علامت خورده است

SLEEPING پردازه ای که به دلیل نداشتن lock امکان اجرا ندارد و منتظر فراخوانده شدن wakeup است تا به حالت RUNNABLE برود.

RUNNABLE در این حالت پردازه اماده به اجرا است و منتظر این است که scheduler آن را انتخاب و به حالت RUNNING در بیاورد.

RUNNING در این حالت پردازه در حال اجرا است.

ZOMBIE وقتی پردازه ی فرزند exit می کند در همان لحظه نمی میرد بلکه به حالت ZOMBIE می رود تا موقعی که پردازه پدر wait را فرا بخواند.

تابع sched() چه وظیفه ای دارد؟

Sched double-checks those conditions (2813-2818) and then an implication of those conditions: since a lock is held, the CPU should be running with interrupts disabled. Finally, sched calls swtch to save the current context in proc->context and switch to the scheduler context in cpu->scheduler.

Sched شروط رها کردن پردازنده توسط پردازه را چک می کند و از آنجا که پردازه باید با غیرفعال بودن وقفه ها کار کند در اخر swtch را فرا می خواند تا با context switch پردازنده را رها کند.

4. تغییری در توابع دسته دوم داده تا تنها پردازه صاحب قفل، قادر به آزادسازی آن باشد.

قفل معادل در هسته لینوکس را به طور مختصر معرفی نمایید.

Sleeping lock که دارای دو فراخوانی سیستمی block() و wakeup(P) که اولی پردازه ای که این فرخوانی سیستم را اجرا کرده متوقف می کند و دومی پردازه ی متوقف شده ی P را اجرا می کند.

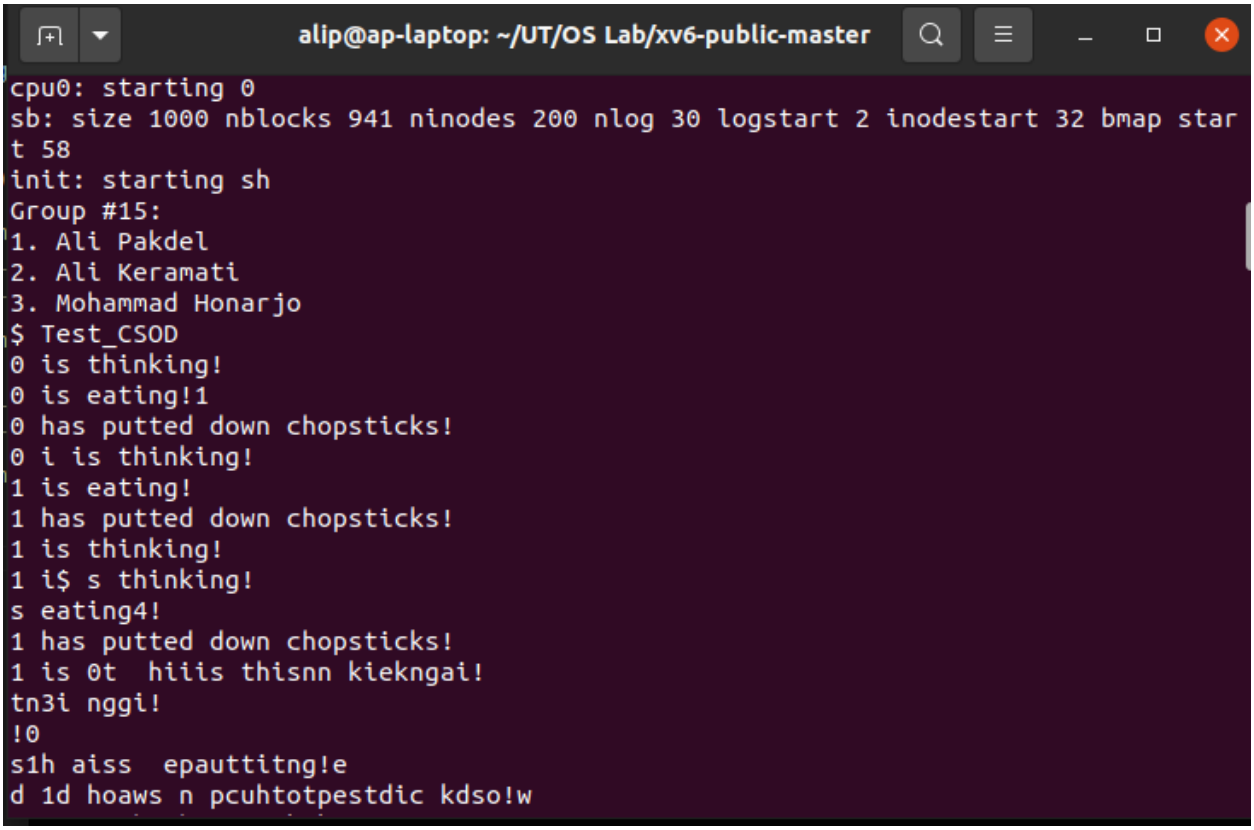
5. یکی از روشهای افزایش کارایی در بارهای کاری چندریسه ای استفاده از حافظه تراکنشی بوده که در کتاب نیز به آن اشاره شده است. به عنوان مثال این فناوری در پردازنده های جدیدتر اینتل تحت عنوان افزونه های همگام سازی تراکنشی (TSX) پشتیبانی میشود. آن را مختصراً شرح داده و نقش حذف قفل را در آن بیان کنید؟

Mainbook page 312

Quite often in computer science, ideas from one area of study can be used to solve problems in other areas. The concept of transactional memory originated in database theory, for example, yet it provides a strategy for process synchronization. A memory transaction is a sequence of memory read –write operations that are atomic. If all operations in a transaction are completed, the memory transaction is committed. Otherwise, the operations must be aborted and rolled back. The benefits of transactional memory can be obtained through features added to a programming language. Consider an example. Suppose we have a function update() that modifies shared data. Traditionally, this function would be written using mutex locks (or semaphores) such as the following: void update () { acquire(); /* modify shared data */ release(); } However, using synchronization mechanisms such as mutex locks and semaphores involves many potential problems, including deadlock. Additionally, as the number of threads increases, traditional locking doesn't scale as well, because the level of contention among threads for lock ownership becomes very high. As an alternative to traditional locking methods, new features that take advantage of transactional memory can be added to a programming language. In our example, suppose we add the construct atomic{S}, which ensures that the operations in S execute as a transaction. This allows us to rewrite the update() function as follows: void update () { atomic { /* modify shared data */ } } The advantage of using such a mechanism rather than locks is that the transactional memory system—not the developer—is responsible for guaranteeing atomicity. Additionally, because no locks are involved, deadlock is not possible. Furthermore, a transactional memory system can identify which statements in atomic blocks can be executed concurrently, such as concurrent read access to a shared variable. It is, of course, possible for a programmer to identify these situations and use reader–writer locks, but the task becomes increasingly difficult as the number of threads within an application grows.

حافظه تراکنشی یا همان Transactional Memory یک دنباله از عملیات های خواندن از حافظه و نوشتن به آن است که atomic هستند. اگر همه ی عملیات ها در یک تراکنش (transaction) کامل انجام شوند حافظه تراکنشی commit شده است در غیر اینصورت عملیات ها باید abort شده و به مرحله ی قبل باز گردد.

مزیت استفاده از این مکانیزم به جای استفاده از lock این است که سیستم حافظه تراکنشی (نه برنامه نویس) وظیفه ی atomic بودن را دارد و همچنین چون lock وجود ندارد وجود deadlock ممکن نیست.


alip@ap-laptop: ~/UT/OS Lab/xv6-public-master
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
Group #15:
1. Ali Pakdel
2. Ali Keramati
3. Mohammad Honarjo
$ Test_CSOD
0 is thinking!
0 is eating!1
0 has putted down chopsticks!
0 i is thinking!
1 is eating!
1 has putted down chopsticks!
1 is thinking!
1 i$ s thinking!
s eating4!
1 has putted down chopsticks!
1 is 0t hiiis thisnn kiekngai!
tn3i nggi!
!0
s1h aiss epauttitng!e
d 1d hoaws n pcuhtotpestdic kdso!w