دانشگاه علم و صنعت

دانشکده مهندسی کامپیوتر

درجه تحصیلی: کارشناسی

گزارشکار تکلیف3 OS

گردآورنده:

پرنیان شاکریان - 99400064

استاد:

دکتر انتظاری

سال تحصیلی: اردیبهشت ۱۴۰۲

خلاصه:

در تکلیف سوم سیستم عامل قصد داریم از lock و semaphore در حل تمارین استفاده کنیم. اگر بخواهیم به طور خلاصه در مورد ماهیت کدها توضیح دهیم Process، پردازه مجموعه‌ای از کدها، حافظه، داده و سایر منابع است. Thread توالی از کدها به حساب می‌آید که در داخل محدوده یک پردازه اجرا می‌شود. lock یک مکانیسم همگام سازی اولیه است که انحصار متقابل برای منابع مشترک را فراهم می‌کند. هنگامی که یک Thread یک lock می‌گیرد، دسترسی انحصاری به منبع پیدا می‌کند و تا زمانی که lock آزاد نشود، از دسترسی سایر Thread ها به آن جلوگیری می‌کند. lock را می توان با استفاده از الگوریتم های مختلفی مانند mutexes پیاده سازی کرد. سمافور یک مکانیسم همگام سازی پیشرفته تر است که به چندین رشته یا فرآیند اجازه می دهد تا به یک منبع مشترک، به صورت همزمان دسترسی داشته باشند. (تا حداکثر تعداد مشخصی) سمافور از یک شمارنده تشکیل شده است که تعداد منابع موجود را ردیابی می کند و مکانیسم همگام سازی و سیگنال‌دهی را برای رشته‌ها فراهم می کند تا منابع را به دست آورند و آزاد کنند.

شرح کلی:

۱. هر رشته دارای یک شی از نوع pthread_t مرتبط با آن است که شناسه آن را می گوید. (نمی تواند توسط چندین رشته به طور همزمان استفاده شود)
۲. وقتی یک رشته نیاز به یک منبع مشترک دارد، ابتدا باید lock یا سمافور را به دست آورد. پس از اتمام کار، باید آنها را آزاد کرده تا رشته‌های دیگر بتوانند به منبع دسترسی پیدا کنند.
۳. تمام کتابخانه ها و headerهای مورد نیاز ما برای تمرین 3 به طور کلی و برای آسانی کار اینجا نوشته شده است.

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
```

سوال ۳: محاسبه‌گرها.

ابتدا کتابخانه‌های مورد نظر خود را include خواهیم کرد. (کتابخانه های مورد نیاز ما برای این تمرین در اول داک نوشته شده است). در شروع کد مقادیر ثابتی موردنظر خود را وارد میکنیم، که حداکثر اندازه صف کار و تعداد رشته‌های ماشین حسابی را که ایجاد خواهند شد را تعیین می کند. سپس یک ساختار Task را تعریف می کنیم تا وظیفه ای که باید توسط یک رشته ماشین حساب انجام شود را نشان داده و شامل دو مقدار ورودی، عملیاتی که باید انجام شود، و تعداد عملیاتی که تاکنون روی این کار انجام شده است، میباشد. سپس آرایه task_queue از ساختارهای Task را برای نگهداری وظایفی که باید انجام شوند ایجاد می کنیم. queue_head و queue_tail نمایانگر شاخص‌های head و tail صف هستند. خط ۱۹ و ۲۰ متغیرهای mutex و شرطی هستند که برای همگام‌سازی رشته هنگام دسترسی به صف استفاده می‌شوند.

تابع enqueue_task یک task را به صف وظایف اضافه می‌کند. ابتدا mutex را قفل کرده تا مطمئن شود که فقط یک رشته می تواند در یک زمان به صف دسترسی داشته باشد. اگر صف از قبل پر باشد، تابع روی متغیر شرط queue_not_empty منتظر می ماند تا یک task از صف حذف شود. هنگامی که فضا در صف وجود دارد، task به دم صف اضافه می شود و شاخص queue_tail به روز می شود. سپس یک سیگنال روی متغیر شرط ارسال می‌شود تا رشته‌های منتظر را فعال کند و قفل mutex باز می‌شود. تابع dequeue_task یک task را از صف وظایف حذف می‌کند. ابتدا mutex را قفل می کند، اگر صف خالی باشد، تابع روی متغیر شرط queue_not_empty منتظر می ماند تا یک task به صف اضافه شود. هنگامی که یک task در صف وجود دارد، از سر صف حذف شده و برگردانده می شود و queue_head به‌روزرسانی می‌شود، سیگنالی روی متغیر شرط ارسال می‌شود تا رشته‌های منتظر را صدا کند، و قفل mutex باز می‌شود. تابعی به نام calculate را تعریف می کنیم که یک ساختار Task را به عنوان پارامتر خود می گیرد و یک ساختار Task را برمی گرداند. اگر «+»، «-»، «*» یا «/» باشد، عملیات حسابی مربوطه را در فیلدهای input1 و input2 انجام می‌دهد و نتیجه را در result ذخیره می‌کند. اگر فیلد عملیات هیچ کدام از اینها نباشد، یک پیغام خطا چاپ می کند و از برنامه خارج می شود. در ادامه تابع فیلد num_operations را بررسی می کند تا

تعیین کند که آیا یک ساختار Task جدید ایجاد شده و آن را در صف قرار میدهد. اگر عملیات '+' یا '-' باشد و num_operations زوج باشد، یک Task جدید با result در input1، input2 و مقادیر num_operations ایجاد می‌کند. اگر عملیات '*' یا '/' باشد و num_operations فرد باشد، یک Task جدید با input1 در فیلد input1 ایجاد کرده، در فیلد input2 نتیجه را می‌گیرد و مقادیر num_operations را به عنوان task اصلی ایجاد می‌کند. اگر هیچ یک از شرط ها برآورده نشود، هیچ کار جدیدی انجام نمیدهد. تابع alculator_thread، thread را اجرا می‌کند. ابتدا، یک شیء از نوع Task ایجاد می‌کند. سپس شناسه thread را از طریق پارامتر arg دریافت و آن را به یک متغیر محلی به نام id تبدیل می‌کند. در بدنه حلقهwhile ، تابع dequeue_task را صدا می‌زند تا یک task از صف را برداشته و آن را در متغیر ذخیره کند. سپس با صدا زدن تابعcalculate محاسبات لازم برای عملیات را انجام داده و نتیجه را در متغیر task ذخیره می‌کند. سپس با استفاده از تابع printf، نتیجه محاسبات را به همراه شناسه و تعداد عملیات‌ها چاپ می‌کند. در پایان، با صدا زدن تابع pthread_exit، thread را خاتمه می‌دهد.

تابع main با اعلام دو آرایه شروع می شود: calculator_threads از نوع pthread_t و num_arr از نوع int. سپس از یک حلقه for برای مقداردهی اولیه num_arr استفاده می کند. سپس یک حلقه for دیگر برای ایجاد رشته با استفاده از pthread_create استفاده می شود. برای هر رشته، تابع calculator_thread به عنوان روال شروع رشته استفاده می شود و یک اشاره گر به عنصری از آرایه num_arr به عنوان آرگومان ارسال می شود. پس از ایجاد تمام رشته‌ها، برنامه وارد یک حلقه بی‌نهایت می‌شود و از کاربر می‌خواهد یک عملیات ریاضی را (به عنوان مثال، «۲ * ۳») وارد کند. ورودی با استفاده از scanf خوانده می‌شود و اگر معتبر باشد، task با استفاده از تابع enqueue_task در task_queue قرار می‌گیرد و اگر ورودی نامعتبر باشد، یک پیام خطا در stderr چاپ می شود. برنامه تا زمانی که به صورت خارجی خاتمه پیدا نکند به حلقه زدن و پذیرش وظایف جدید ادامه می دهد. در نهایت تابع 0 main را برمی گرداند که نشان دهنده اجرای موفق برنامه است.

خروجی های زیر را برای ورودی های دلخواه من خواهیم داشت:

Output Clear

```
/tmp/WTC0t303zs.o
please Enter the task: 2-1
please Enter the task: Calculator 140730499390976: 1 - 0 = 1, num_operations=1
5+4
please Enter the task: Calculator 140730499390984: 9 + 0 = 9, num_operations=1
8-10
please Enter the task: Calculator 140730499390992: -2 - 0 = -2, num_operations=1
4*3
please Enter the task: Calculator 140730499390976: 12 * 0 = 12, num_operations=1
Calculator 140730499390976: 48 * 0 = 48, num_operations=2
8/4
please Enter the task: Calculator 140730499390980: 2 / 0 = 2, num_operations=1
Calculator 140730499390980: 4 / 0 = 4, num_operations=2
8/3
please Enter the task: Calculator 140730499390984: 2.66667 / 0 = 2.66667,
    num_operations=1
Calculator 140730499390984: 3 / 0 = 3, num_operations=2
please Enter the task (input1 operation input2): 3-1

please Enter the task (input1 operation input2): Calculator 140726993887920: 2 - 0 = 2
    , num_operations=1
5+6
please Enter the task (input1 operation input2): Calculator 140726993887928: 11 + 0 =
    11, num_operations=1
11/3
please Enter the task (input1 operation input2): Calculator 140726993887936: 3.66667 /
    0 = 3.66667, num_operations=1
Calculator 140726993887924: 3 / 0 = 3, num_operations=2
12/3
please Enter the task (input1 operation input2): Calculator 140726993887932: 4 / 0 = 4
    , num_operations=1
Calculator 140726993887932: 3 / 0 = 3, num_operations=2
5*5
please Enter the task (input1 operation input2): Calculator 140726993887924: 25 * 0 =
    25, num_operations=1
```