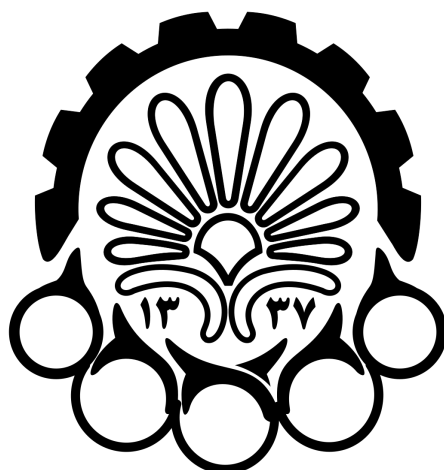


سیستم‌های عامل
دکتر جوادی



دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)
دانشکده مهندسی کامپیوتر

رضا آدینه پور ۴۰۲۱۳۱۰۵۵

تمرین سری دوم

۲۴ آبان ۱۴۰۲

سوال اول

فرض کنید برنامه ای داریم که با چند نخ در حال اجرا می باشد، با توجه به این موضوع درستی و نادرستی عبارات زیر را با ذکر دلیل مشخص کنید و توضیح دهید.

۱. اگر یک نخ آرگومان های خاصی را به یک تابع در برنامه ارسال کند، این آرگومان ها برای نخ های دیگر قابل مشاهده هست
۲. اگر یک نخ با استفاده از دستور malloc حافظه اضافی به خود تخصیص دهد می تواند باعث به وجود آمدن خطای out of memory برای نخ دیگری در برنامه شود
۳. رشته های سطح کاربر توسط کتابخانه سطح کاربر برنامه ریزی می شوند و بدون اینکه هسته از عملیات آن ها مطلع باشد کار می کنند
۴. زمان context switch در رابطه با نخ های سطح هسته کمتر طول می کشد.
۵. نخ های سطح کاربر نمی توانند به صورت موازی واقعی true parallelism در سیستم های چند هسته ای اجرا شوند زیرا توسط یک رشته در سطح هسته مدیریت می شوند.

پاسخ

۱. درست
توضیحات: در برنامه‌نویسی چندنخی، هر نخ دارای محیط اجرایی خود است و برخی متغیرها، مانند آرگومان‌های تابع، در حافظه اشتراکی میان نخ‌ها قرار می‌گیرند. بنابراین، اگر یک نخ آرگومانی را تغییر دهد، تغییرات ممکن است بر روی نخ‌های دیگر تأثیر بگذارد و آنها نیز مقدار تغییر یافته را مشاهده کنند

۲. نادرست
توضیحات: هر نخ دارای منطقه‌ای از حافظه برای استفاده خود است. اگر یک نخ بیش از حد حافظه را تخصیص دهد، حافظه موجود برای سایر نخ‌ها کاهش می‌یابد. این ممکن است منجر به خطای out of memory برای نخ‌های دیگری شود که نتوانند حافظه لازم را برای ادامه اجرا تخصیص دهند.

۳. درست
توضیحات: رشته‌های سطح کاربر توسط کتابخانه‌های سطح کاربر برنامه‌ریزی می‌شوند و اجرای آنها توسط هسته انجام نمی‌شود. این رشته‌ها به صورت مستقل و به طور همزمان اجرا می‌شوند و هسته از وجود و عملکرد آنها آگاهی ندارد

۴. نادرست
توضیحات: زمان context switch بین نخ‌های سطح هسته Kernel Level Thread بیشتر از زمان switch context بین نخ‌های سطح کاربر است. زیرا در context switch بین نخ‌های سطح هسته، وضعیت کامل نخ، از جمله مجموعه ثبات نخ و محتوای ثبات شده، باید ذخیره شود و سپس وضعیت جدید نخ بارگذاری شود. این عملیات بیشترین زمان را می‌طلبد. در حالی که در context switch بین نخ‌های سطح کاربر، زمان کوتاهی برای ذخیره و بارگذاری وضعیت نخ‌ها صرف می‌شود، زیرا این نخ‌ها توسط کتابخانه سطح کاربر برنامه‌ریزی مدیریت می‌شوند و اطلاعات کمتری برای تغییر وضعیت آنها نیاز است

سوال دوم

خروجی قطعه کد زیر را پیش بینی کنید و پیش بینی خود را توضیح دهید. سپس به سوالات زیر پاسخ دهید.

Listing 1: Some Code

```
#define NUM_THREADS 3
int shared_value = 10

void* increaseValue(void* threadID)
{
    long tid = (long)threadID;
    printf("Thread%id:shared_value_before_increament:%d\n", tid, shared_value);
    shared_value += 5;
    printf("Thread%id:shared_value_after_increament:%d\n", tid, shared_value);
    pthread_exit(NULL);
}

int main()
{
    pthread_t threads [NUM_THREADS];
    long t;
    for(t=0; t<NUM_THREADS; t++)
    {
        printf("creating_thread_%id\n", t);
        pthread_create(&threads[t], NULL, increaseValue, (void*)t);
    }

    for(t=0; t<NUM_THREADS; t++)
    {
        pthread_join(threads[t], NULL);
    }

    printf("All_threads_have_completed._Final_shared_value:_%d\n", shared_value);
    pthread_exit(NULL);

    return 0;
}
```

پاسخ

این برنامه، یک برنامه چندنخی است که از کتابخانه pthread برای ایجاد و مدیریت نخ‌ها استفاده می‌کند. برنامه شامل تابع `increaseValue` است که توسط هر نخ صدا زده می‌شود. هر نخ درون تابع `increaseValue` مقدار مشترک `shared_value` را افزایش می‌دهد. سپس در تابع `main`، نخ‌ها ایجاد و به صورت موازی اجرا می‌شوند. پس از اتمام هر نخ، با استفاده از تابع `pthread_join`، منتظر تمام شدن هر نخ می‌ماند و در نهایت مقدار نهایی `shared_value` چاپ می‌شود.

۱. در صورتی که دو تابع `pthread_create` و `pthread_join` در یک حلقه صدا زده شوند. خروجی برنامه چه خواهد بود و علت آن را توضیح دهید

توضیحات: در صورتی که توابع `pthread_create` و `pthread_join` در یک حلقه صدا زده شوند، خروجی برنامه نامعلوم است. زیرا عملکرد توابع مذکور بستگی به زمانبندی سیستم عامل و زمان اجرای هر نخ دارد. وقتی توابع در یک حلقه صدا زده می‌شوند، ترتیب شروع و اجرای نخ‌ها ممکن است تغییر کند و به همین دلیل خروجی برنامه قابل پیش‌بینی نیست. بنابراین، مقدار `shared_value` در نهایت ممکن است نامعلوم باشد

۲. تفاوت بین استفاده از رشته‌ها و استفاده از `fork` برای رسیدن به موازی سازی را توضیح دهید.
توضیحات: تفاوت استفاده از `Threads` و استفاده از `fork` برای موازی‌سازی به شرح زیر است:

(آ) **Threads:** در برنامه‌نویسی چندنخی با استفاده از رشته‌ها، پردازش موازی در یک فرآیند انجام می‌شود. تمام رشته‌ها در یک فضای آدرسی مشترک قرار دارند و می‌توانند به طور مستقیم به منابع و متغیرهای مشترک دسترسی داشته باشند. این باعث اشتراک منابع و ارتباط آسان بین رشته‌ها می‌شود. با ایجاد رشته‌ها، هزینه ایجاد و تغییر حالت آنها کمتر از هزینه ایجاد و تغییر حالت فرآیندها است

(ب) **fork:** در زمان استفاده از `fork` برای موازی‌سازی، یک فرآیند اصلی و یک فرآیند جدید ایجاد می‌شود. هر فرآیند جدید دارای یک فضای آدرسی مستقل است و هیچ اشتراکی با فرآیند اصلی ندارد. برای ارتباط بین فرآیندها، مکانیزم‌هایی مانند لوله‌ها (`pipes`) یا صف‌صفحه‌های پیوندی (`shared memory`) بین فرآیندها بکار می‌روند. هزینه ایجاد و تغییر حالت فرآیندها بیشتر از هزینه رشته‌ها است

۳. آیا چند نخ می‌توانند تابع `pthread_join` را بر روی یک نخ هدف فراخوانی کنند یا محدودیتی وجود دارد که تنها یک نخ انتظارها را برای هر نخ هدف می‌تواند داشته باشد؟

توضیحات: در برنامه‌های چندنخی، تابع `pthread_join` برای انتظار تمام شدن یک نخ هدف استفاده می‌شود. هر نخ می‌تواند تابع `pthread_join` را بر روی هر نخ هدف فراخوانی کند و منتظر تمام شدن آن نخ باشد. بنابراین، با توجه به سوال شما، می‌توان تعدادی نخ تابع `pthread_join` را بر روی یک نخ هدف فراخوانی کنند و هرکدام از این نخ‌ها منتظر تمام شدن آن نخ هدف خواهند بود. محدودیتی در تعداد نخ‌هایی که می‌توانند تابع `pthread_join` را بر روی یک نخ هدف فراخوانی کنند وجود ندارد

سوال سوم

با توجه به برنامه زیر به موارد خواسته شده پاسخ دهید. (فرض کنید Thread ها در ابتدا به ترتیب شروع می‌شوند)

Listing 2: Some Code

P1:

```
fork()
func1:
    print("x")
func2:
    fork();
t1 = pthread(func1)
t2 = pthread(func2)
t3 = pthread(func1)
join t1, t2, t3
wait()
```

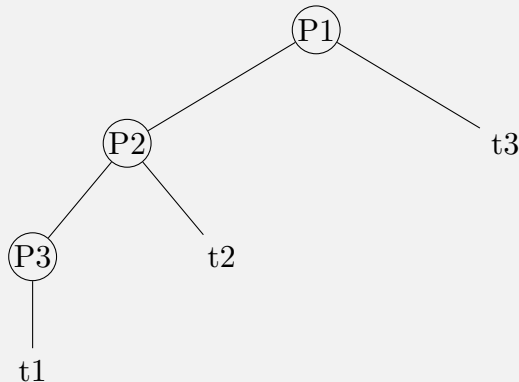
پاسخ

۱. تعداد x های چاپ شده بعد از اتمام برنامه:
تابع func1 به صورت موازی توسط ۳ نخ فراخوانی می‌شود. پس در مجموع، این تابع ۳ بار اجرا می‌شود و در نتیجه ۳ بار "x" چاپ می‌شود.

۲. تعداد پردازش‌ها بعد از اتمام برنامه:
در ابتدا، یک پردازش وجود دارد. با اجرای دستور fork()، یک پردازش جدید ایجاد می‌شود و تعداد پردازش‌ها به ۲ افزایش می‌یابد. در تابع func2، یک fork() دیگر وجود دارد که تعداد پردازش‌ها به ۴ افزایش می‌یابد. پس از اتمام تمامی thread ها، تعداد پردازش‌ها ۴ است.

۳. تعداد ترد ها بعد از اتمام برنامه:
در ابتدا، یک نخ اصلی وجود دارد. سپس با ایجاد تردهای t1، t2 و t3، تعداد تردها به ۴ افزایش می‌یابد. پس از اتمام تردها، تعداد تردها ۴ است.

۴. رسم درخت پردازش‌ها و مشخص کردن تعداد ترد های هرکدام:
پردازش اصلی P دارای ۳ ترد است: t1، t2، t3. پردازش اولیه که از طریق fork ایجاد شد دارای ۱ ترد است: t2. پردازش‌های دوم و سوم که از طریق fork در تابع func1 ایجاد شدند هر کدام دارای ۱ ترد است: t1 و t3.



۵. خلاصه از فرآیند را شرح دهید:

- در ابتدا، یک پردازش ایجاد می‌شود.
- با اجرای دستور fork()، یک پردازش جدید ایجاد می‌شود و تعداد پردازش‌ها به ۲ افزایش می‌یابد.
- در پردازش اصلی، ۳ ترد (t1، t2 و t3) با استفاده از تابع pthread ایجاد می‌شوند.
- در پردازش اصلی، ترد t1 تابع func1 را فراخوانی می‌کند و یک "x" چاپ می‌شود.
- در پردازش اصلی، ترد t2 تابع func2 را فراخوانی می‌کند و یک پردازش جدید ایجاد می‌شود که نام آن همچنان P است.
- در پردازش جدید (P)، ترد t2 تابع func2 را فراخوانی می‌کند و یک پردازش جدید ایجاد می‌شود.
- در پردازش جدید (P)، ترد t1 تابع func1 را فراخوانی می‌کند و یک "x" چاپ می‌شود.
- در پردازش اصلی، ترد t3 تابع func1 را فراخوانی می‌کند و یک "x" چاپ می‌شود.
- پس از اتمام تمامی تردها، تعداد پردازش‌ها ۴ است و برنامه به پایان می‌رسد.

سوال چهارم

با فرض موفقیت آمیز بودن اجرای دستورات `fork` و `execv` خروجی قطعه کد زیر را به صورت دقیق و با ذکر دلیل بیان کنید.

Listing 3: Some Code

```
int main()
{
    pid_t pid;
    pid = fork();
    if(pid == 0)
    {
        printf("process_1\n");
        char* args[] = {"ls", "-l", NULL};
        execv("/bin/ls", args);
        printf("process_1_finished\n");
    }
    else if(pid > 0)
    {
        printf("process_2\n");
        wait(NULL);
        printf("process_1_terminated\n");
    }
    return 0;
}
```


پاسخ

کد فوق یک برنامه ساده را نشان می‌دهد که از تابع `fork` برای ایجاد یک فرزند جدید استفاده می‌کند. در ادامه، فرزند ایجاد شده با استفاده از تابع `execv` اجرای برنامه `"ls"` را انجام می‌دهد. در ادامه، والد منتظر فرزند خود می‌ماند تا اجرای فرزند به پایان برسد. سپس والد پیامی چاپ کرده و برنامه پایان می‌یابد. حال به صورت دقیق خروجی برنامه را تحلیل می‌کنیم:

۱. در صورت موفقیت آمیز بودن تابع `fork`، والد یک پردازش فرزند را ایجاد می‌کند و بازگشتی غیر صفر دارد. در صورتی که خطایی رخ دهد، بازگشتی کمتر از صفر خواهد داشت.

۲. در صورتی که `pid` برابر ۰ باشد، به این معنی است که کد در حال اجرا در پردازش فرزند است. در این حالت:

- `process 1` چاپ می‌شود.
- یک آرایه از رشته‌ها به نام `args` تعریف می‌شود که مسیر برنامه `"ls"` و پارامترهای آن را مشخص می‌کند.
- تابع `execv` فراخوانی می‌شود تا برنامه `"ls"` را با استفاده از آرگومان‌های مشخص شده اجرا کند. اگر این تابع با موفقیت اجرا شود، کنترل برنامه به برنامه `"ls"` منتقل می‌شود و دستورات بعدی در کد اجرا نمی‌شوند.
- در صورتی که تابع `execv` با خطا مواجه شود و اجرای برنامه `"ls"` انجام نشود، پیام `1 finished` چاپ می‌شود. این پیام هیچگاه نمایش داده نمی‌شود زیرا کنترل برنامه به برنامه `"ls"` منتقل می‌شود و دستورات بعدی اجرا نمی‌شوند.

۳. در صورتی که `pid` بزرگتر از ۰ باشد، به این معنی است که کد در حال اجرا در پردازش والد است. در این حالت:

- `process 2` چاپ می‌شود.
- با استفاده از تابع `wait`، والد منتظر اجرای فرزند خود می‌ماند تا به پایان برسد.
- پس از اتمام اجرای فرزند، پیام `process 1 terminated` چاپ می‌شود.

بنابراین، خروجی نهایی برنامه فوق، به ترتیب چاپ شدن پیام در این کد، در صورت موفقیت آمیز بودن تابع `fork`، یک پردازش فرزند ایجاد می‌شود و بازگشتی غیر صفر دارد. اگر `pid` برابر ۰ باشد، به این معنی است که کد در حال اجرا در پردازش فرزند است. در این حالت، `Process 1` چاپ می‌شود و سپس تابع `execv` فراخوانی می‌شود تا برنامه `"ls"` را با استفاده از آرگومان‌های مشخص شده اجرا کند. اگر این تابع با موفقیت اجرا شود، کنترل برنامه به برنامه `"ls"` منتقل می‌شود و دستورات بعدی در کد اجرا نمی‌شوند. اگر تابع `execv` با خطا مواجه شود و اجرای برنامه `"ls"` انجام نشود، پیام `process 1 finished` چاپ می‌شود.

در صورتی که `pid` بزرگتر از ۰ باشد، به این معنی است که کد در حال اجرا در پردازش والد است. در این حالت، `process 2` چاپ می‌شود و با استفاده از تابع `wait`، والد منتظر اجرای فرزند خود می‌ماند تا به پایان برسد. پس از اتمام اجرای فرزند، پیام `process 1 terminated` چاپ می‌شود.

در نتیجه، خروجی برنامه به صورت زیر خواهد بود:

```
process 2
process 1
<ls results of program>
process 1 terminated
```

توجه کنید که خروجی برنامه `"ls"` به عنوان نتیجه اجرای `execv` وابسته به محتوای فایل‌ها و دایرکتوری‌های حاضر در مسیر `bin/ls/` است.

سوال پنجم

با فرض آنکه پردازش‌های producer و consumer به نحو زیر پیاده‌سازی شده‌اند. اگر سایز buffer برابر با ۵ باشد، و متغیرهای in و out برابر با صفر باشد، در هر مورد، خروجی را با ذکر دلیل مشخص کنید.

Listing 4: Some Code

Producer :

```
int next_produced = 0;
while(next_produced < 10)
{
    buffer[in] = ++next_produced;
    in = (in + 1) % BUFFER_SIZE
}
```

Listing 5: Some Code

Consumer :

```
int next_consumed, sum;
while(next_consumed < 10)
{
    if(in == out)
        continue;
    next_consumed = buffer[out];
    out = (out + 1) % BUFFER_SIZE
    sum += next_consumed;
}
printf("%d", sum);
```

پاسخ

۱. به ازای هر یک بار اجرای بدنه، حلقه producer یک بار بدنه حلقه consumer اجرا شود. توضیحات: اگر تنها یک بار حلقه بدنه مصرف‌کننده و یک بار حلقه بدنه تولیدکننده اجرا شوند، مقدار sum برابر با صفر خواهد بود. زیرا تولیدکننده ۱۰ عدد از ۱ تا ۱۰ را در بافر قرار می‌دهد، اما مصرف‌کننده هیچ عددی را مصرف نمی‌کند. این اتفاق به دلیل این است که بعد از قرار دادن اعداد در بافر، مصرف‌کننده هنوز شروع به مصرف نکرده است و در شرایطی که in و out برابر باشند، حلقه مصرف‌کننده به continue می‌رود و از مصرف عدد خودداری می‌کند.

۲. به ازای هر دو بار اجرای بدنه، حلقه producer یک بار بدنه حلقه consumer اجرا شود. توضیحات: اگر دو بار حلقه بدنه مصرف‌کننده و یک بار حلقه بدنه تولیدکننده اجرا شوند در این حالت، مقدار sum برابر با ۵ خواهد بود. تولیدکننده ۱۰ عدد از ۱ تا ۱۰ را در بافر قرار می‌دهد و مصرف‌کننده در دو بار اجرا، پنج عدد از بافر را مصرف می‌کند. در هر بار اجرا، مصرف‌کننده با بررسی شرط $out == in$ به مصرف عدد پرداخته و مقدار next_consumed را در sum اضافه می‌کند. اما در بین دو بار اجرا، مقدار in و out به هم نزدیک می‌شوند و همیشه شرط $out == in$ برقرار نمی‌شود. به عبارت دیگر، مصرف‌کننده در هر بار اجرا در حداکثر یک عدد را مصرف می‌کند و در بارهای بعدی باید منتظر تولیدکننده بماند. بنابراین، مقدار sum برابر با جمع اعداد ۱ تا ۵ خواهد بود که برابر با ۵ است.

۳. به ازای هر سه بار اجرای بدنه، حلقه producer یک بار بدنه حلقه consumer اجرا شود. توضیحات: اگر سه بار حلقه بدنه مصرف‌کننده و یک بار حلقه بدنه تولیدکننده اجرا شوند در این حالت، مقدار sum برابر با ۱۵ خواهد بود. تولیدکننده ۱۰ عدد از ۱ تا ۱۰ را در بافر قرار می‌دهد و مصرف‌کننده در سه بار اجرا، همه اعداد موجود در بافر را مصرف می‌کند. در اولین بار اجرا، مصرف‌کننده با بررسی شرط $out == in$ به مصرف اعداد ۱ تا ۵ می‌پردازد و مقدار sum را به ترتیب با ۱، ۲، ۳، ۴ و ۵ افزایش می‌دهد.

سوال ششم

تا به اینجای درس با موازی سازی فرایندها آشنا شده اید. یکی از مفاهیم در این خصوص multi-core است. برای مشاهده مشخصات پردازنده می‌توان از دستور `lscpu` استفاده کرد. این دستور، اطلاعات پردازنده را از فایل `proc/cpuinfo` می‌خواند و در ساختاری مناسب به نمایش می‌گذارد. خروجی دستور `lscpu` را در گزارش خود قرار دهید.

پاسخ

```

+ - lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Address sizes:          39 bits physical, 48 bits virtual
Byte Order:             Little Endian
CPU(s):                 12
On-line CPU(s) list:   0-11
Vendor ID:              GenuineIntel
Model name:             Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
CPU family:             6
Model:                 158
Thread(s) per core:    2
Core(s) per socket:    6
Socket(s):              1
Stepping:               10
CPU(s) scaling MHz:    20%
CPU max MHz:            4100.0000
CPU min MHz:            800.0000
BogoMIPS:               4399.99
Flags:                  fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mc
a cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss
ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art
arch_perfmon pebs bts rep_good nopl xtopology nonstop
tsc cpuid aperfperf pni pclmuldq dtes64 monitor ds_cp
l vmx est tm2 sse3 sdbg fma cx16 xtpr pcdm pcid sse4_1
sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave
avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault
epb invpcid_single pti ssbd ibrs ibpb stibp tpr_shadow
flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1
avx2 smep bmi2 erms invpcid mpx rdseed adx snap clflush
opt intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida
arat pln pts hwp hwp_notify hwp_act_window hwp_epp vnm
md_clear flush_lid arch_capabilities

Virtualization features:
Virtualization:         VT-x
Caches (sum of all):
L1d:                    192 KiB (6 instances)
L1i:                    192 KiB (6 instances)
L2:                     1.5 MiB (6 instances)
L3:                     9 MiB (1 instance)
NUMA:
NUMA node(s):           1
NUMA nodes CPU(s):     0-11
Vulnerabilities:
Gather data sampling:   Mitigation; Microcode
Itlb multihit:          KVM: Mitigation: VMX disabled
L1tf:                   Mitigation; PTE Inversion; VMX conditional cache flushes
SMT vulnerable:

```

دستور `top` یکی از دستورات کاربردی در مدیریت پردازنده هاست. این دستور اطلاعات مربوط به `thread` های پردازنده، شماره پردازنده، درصد استفاده از هر هسته پردازنده، میزان استفاده هر پردازنده از کل `cpu` و ... را به ما نشان می‌دهد.

این دستور به صورت پیش فرض همه اطلاعات درمورد پردازنده ها را به ما نمایش نمی دهد اما می‌توانیم به صورت دستی خودمان یک سری از اطلاعاتی که نیاز داریم را به ستون های اطلاعاتش اضافه و کم کنیم. برای اینکار پس از اجرای `top` دکمه `f` را فشار دهید. صفحه ای مشابه تصویر زیر برایتان نمایش داده میشود. با کلیدهای بالا و پایین کیبورد می‌توان در این گزینه‌ها جا به جا شد و با کلید `space` می‌توان گزینه‌های مختلفی را انتخاب کرد.

پاسخ

```

top
Fields Management for window 1:Def1, whose current sort field is %CPU
Navigate with Up/Dn, Right selects for move then <Enter> or Left commits,
'd' or <Space> toggles display, 's' sets sort. Use 'q' or <Esc> to end!

* PID      = Process Id
* USER     = Effective User Name
* PR       = Priority
* NI       = Nice Value
* VIRT     = Virtual Image (KiB)
* RES      = Resident Size (KiB)
* SHR      = Shared Memory (KiB)
* S        = Process Status
* %CPU     = CPU Usage
* %MEM     = Memory Usage (RES)
* TIME+    = CPU Time, hundredths
* COMMAND  = Command Name/Line
PPID      = Parent Process pid
UID       = Effective User Id
RUID      = Real User Id
RUSER     = Real User Name
SUID      = Saved User Id
SUSER     = Saved User Name
GID       = Group Id
GROUP     = Group Name
PGRP      = Process Group Id
TTY       = Controlling Tty
TPGID     = Tty Process Grp Id
SID       = Session Id
nTH       = Number of Threads
P         = Last Used Cpu (SMP)
TIME      = CPU Time
SMAP      = Swapped Size (KiB)
CODE      = Code Size (KiB)
DATA      = Data+Stack (KiB)
nMaj      = Major Page Faults
nMin      = Minor Page Faults
nDRT      = Dirty Pages Count
WCHAN     = Sleeping in Function
Flags     = Task Flags <sched.h>
CGROUPS   = Control Groups
SUPRGIDS  = Supp Groups Ids
SUPGRPS   = Supp Groups Names
TGID      = Thread Group Id
OOMa      = OOMEM Adjustment
OOMs      = OOMEM Score current
ENVIRON    = Environment vars
vMj       = Major Faults delta
VMn       = Minor Faults delta
USED      = Res+Swap Size (KiB)
nsIPC     = IPC namespace Inode
nsMNT     = MNT namespace Inode
nsNET     = NET namespace Inode
nsPID     = PID namespace Inode
nsUSER    = USER namespace Inode
nsUTS     = UTS namespace Inode
LXC       = LXC container name
Rsan      = RES Anonymous (KiB)
RSfd      = RES File-based (KiB)
RSlk      = RES Locked (KiB)
RSsh      = RES Shared (KiB)
CGNAME    = Control Group name
NU        = Last Used NUMA node
LOGID     = Login User Id
EXE       = Executable Path
RSS       = Res Mem (snaps), KiB
PSS       = Proportion RSS, KiB
PSan      = Proportion Anon, KiB
PSfd      = Proportion File, KiB
PSsh      = Proportion Shrd, KiB
USS       = Unique RSS, KiB
ioR       = I/O Bytes Read
ioRop     = I/O Read Operations
ioW       = I/O Bytes Written
ioWop     = I/O Write Operations
AGID      = Autogroup Identifier
AGNI      = Autogroup Nice Value
STARTED   = Start Time from boot
ELAPSED    = Elapsed Running Time
%CUU      = CPU Utilization
%CUC      = Utilization + child
nsCGROUP  = CGRP namespace Inode
nsTIME    = TIME namespace Inode

```

nTH تعداد thread های یک پردازش و p شماره آخرین هسته پردازنده ای که برای پردازش استفاده شده است را نمایش می دهد. این دو گزینه را انتخاب کنید و خروجی آن را در گزارشتان بیاورید.

پاسخ

```

top
top - 13:43:33 up 4:57, 2 users, load average: 0.69, 0.71, 0.78
Tasks: 340 total, 2 running, 338 sleeping, 0 stopped, 0 zombie
%Cpu(s): 3.7 us, 0.9 sy, 0.0 ni, 95.4 id, 0.0 wa, 0.0 hi, 0.1 si, 0.0 st
MiB Mem : 11801.6 total, 560.8 free, 5382.6 used, 7125.8 buff/cache
MiB Swap: 4096.0 total, 4096.0 free, 0.0 used, 6418.9 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM    TIME+  COMMAND      nTH  P
 3308 reza    20   0 5764632 421740 198768 R 30.2  3.5 54:03.58 gnome-shell      32  5
 7080 reza    20   0 2598820 698572 236052 S   6.0  5.8 13:52.98 telegram-deskto    80  8
28444 reza    20   0 716520  70736  55700 S   5.0  0.6  0:08.76 gnome-terminal-     6  7
21826 reza    20   0 2787448 246896 109896 S   2.7  2.0  6:49.95 Isolated Web Co    26 11
4835 reza    20   0 12.4g 904076 250004 S   2.0  7.5 25:25.75 firefox            165  0
6074 reza    20   0 18.8g 240016 111168 S   2.0  2.0  5:18.73 Isolated Web Co    36 11
13350 reza    20   0 2357532 135952 103344 S   2.0  1.1 2:29.22 nekoray             23  1
28516 reza    20   0 15876  6400  4096 R   1.0  0.1  0:00.50 top                  1  9
25502 root    20   0 0 0 0 I   0.7  0.0 0:19.72 kworker/3:2-1915-unordered 1  3
 35 root    20   0 0 0 0 S   0.3  0.0 0:25.95 ksoftirqd/3         1  3
1466 root    0 -20 0 0 0 I   0.3  0.0 1:09.50 kworker/u25:1-1915_flip 1  1
1605 systemd+ 20   0 17060  7564  6668 S   0.3  0.1 0:24.80 systemd-oomd        1  9
3056 reza    9 -11 132784 15456  8612 S   0.3  0.1 0:20.27 pipewire            3  1
3084 reza    20   0 11360  7208  4680 S   0.3  0.1 0:07.77 dbus-daemon        1  1
5174 reza    20   0 18.7g 155972 84432 S   0.3  1.3 4:25.07 WebExtensions      26 10
10779 reza    20   0 2639752 234028 99976 S   0.3  1.9 0:15.69 Isolated Web Co    27 10
12222 reza    20   0 3091876 201268 105864 S   0.3  1.7 0:23.60 nautilus             29 10
13368 reza    20   0 1255468 32188 19628 S   0.3  0.3 0:44.62 nekoray_core        14  7
20825 reza    20   0 2629820 194840 98428 S   0.3  1.6 0:26.90 Isolated Web Co    26  1
22368 root    20   0 0 0 0 I   0.3  0.0 0:01.24 kworker/1:0-events   1  1
27798 root    20   0 0 0 0 I   0.3  0.0 0:00.22 kworker/u24:64-1915 1  1
 1 root    20   0 171492 15476  9144 S   0.0  0.1 0:07.23 systemd             1  0
 2 root    20   0 0 0 0 S   0.0  0.0 0:00.01 kthreadd            1  5
 3 root    0 -20 0 0 0 I   0.0  0.0 0:00.00 rcu_gp              1  0
 4 root    0 -20 0 0 0 I   0.0  0.0 0:00.00 rcu_par_gp          1  0
 5 root    0 -20 0 0 0 I   0.0  0.0 0:00.00 slub_flushwq        1  0
 6 root    0 -20 0 0 0 I   0.0  0.0 0:00.00 netns               1  0
 8 root    0 -20 0 0 0 I   0.0  0.0 0:00.00 kworker/0:0H-events_highpri 1  0
11 root    0 -20 0 0 0 I   0.0  0.0 0:00.00 mm_percpu_wq        1  0
12 root    20   0 0 0 0 I   0.0  0.0 0:00.00 rcu_tasks_kthread   1  1
13 root    20   0 0 0 0 I   0.0  0.0 0:00.00 rcu_tasks_rude_kthread 1  0
14 root    20   0 0 0 0 I   0.0  0.0 0:00.00 rcu_tasks_trace_kthread 1  6
15 root    20   0 0 0 0 S   0.0  0.0 0:00.46 ksoftirqd/0         1  0
16 root    20   0 0 0 0 I   0.0  0.0 0:11.66 rcu_preempt         1  1
17 root    rt   0 0 0 0 S   0.0  0.0 0:00.07 migration/0       1  0
18 root    -51  0 0 0 0 S   0.0  0.0 0:00.00 idle_inject/0       1  0
19 root    20   0 0 0 0 S   0.0  0.0 0:00.01 cpuhp/0             1  0
20 root    20   0 0 0 0 S   0.0  0.0 0:00.00 cpuhp/1             1  1
21 root    -51  0 0 0 0 S   0.0  0.0 0:00.00 idle_inject/1       1  1
22 root    rt   0 0 0 0 S   0.0  0.0 0:00.25 migration/1       1  1

```

همچنین با فشار دادن کلید ۱ در پردازش top می توان میزان استفاده از هر کدام از core های پردازش را مشاهده کرد.

پاسخ

top

```

top - 13:45:21 up 4:59, 2 users, load average: 0.84, 0.75, 0.79
Tasks: 345 total, 1 running, 344 sleeping, 0 stopped, 0 zombie
%Cpu0 : 0.7 us, 0.3 sy, 0.0 ni, 99.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu1 : 0.3 us, 0.7 sy, 0.0 ni, 99.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu2 : 1.3 us, 2.3 sy, 0.0 ni, 96.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu3 : 0.0 us, 0.3 sy, 0.0 ni, 99.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu4 : 2.3 us, 1.7 sy, 0.0 ni, 96.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu5 : 0.0 us, 0.0 sy, 0.0 ni, 100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu6 : 1.0 us, 0.3 sy, 0.0 ni, 98.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu7 : 0.3 us, 0.0 sy, 0.0 ni, 99.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu8 : 21.4 us, 3.1 sy, 0.0 ni, 75.6 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu9 : 0.0 us, 0.0 sy, 0.0 ni, 100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu10 : 0.3 us, 0.7 sy, 0.0 ni, 99.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu11 : 1.7 us, 0.7 sy, 0.0 ni, 97.3 id, 0.0 wa, 0.0 hi, 0.3 si, 0.0 st
MiB Mem : 11801.6 total, 659.2 free, 5384.7 used, 7003.5 buff/cache
MiB Swap: 4096.0 total, 4096.0 free, 0.0 used, 6416.9 avail Mem

```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND	nTH	P
3308	reza	20	0	5739864	422096	198896	S	28.2	3.5	54:35.58	gnome-shell	31	7
7080	reza	20	0	2669600	705760	236852	S	2.7	5.8	13:55.40	telegram-desktop	86	11
21826	reza	20	0	2787448	247152	109896	S	2.7	2.0	6:52.54	Isolated Web Co	26	3
4835	reza	20	0	12.4g	943648	250004	S	1.7	7.8	25:27.97	firefox	165	2
13350	reza	20	0	2357532	135952	103344	S	1.3	1.1	2:30.61	nekoray	23	7
6074	reza	20	0	18.8g	243160	111168	S	1.0	2.0	5:20.16	Isolated Web Co	36	5
25502	root	20	0	0	0	0	I	1.0	0.0	0:20.67	kworker/3:2-1915-unordered	1	3
16	root	20	0	0	0	0	I	0.3	0.0	0:11.76	rcu_preempt	1	4
1466	root	0	-20	0	0	0	I	0.3	0.0	1:10.14	kworker/u25:1-1915_flip	1	4
2044	root	20	0	1947272	43508	30976	S	0.3	0.4	0:17.86	containerd	17	11
4286	root	20	0	541816	38096	29928	S	0.3	0.3	0:04.85	fwupd	6	4
5174	reza	20	0	18.7g	148236	84432	S	0.3	1.2	4:25.88	WebExtensions	26	4
13368	reza	20	0	1255468	31608	19628	S	0.3	0.3	0:44.97	nekoray_core	14	7
22331	reza	20	0	2495776	141740	90116	S	0.3	1.2	0:04.60	Isolated Web Co	26	1
28516	reza	20	0	15876	6400	4096	R	0.3	0.1	0:05.38	top	1	1
1	root	20	0	171492	15476	9144	S	0.0	0.1	0:07.23	systemd	1	2
2	root	20	0	0	0	0	S	0.0	0.0	0:00.01	kthreadd	1	1
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp	1	0
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_par_gp	1	0
5	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	slub_flushwq	1	0
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	netns	1	0
8	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/8:0H-events_highpri	1	0
11	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_percpu_wq	1	0
12	root	20	0	0	0	0	I	0.0	0.0	0:00.00	rcu_tasks_kthread	1	1
13	root	20	0	0	0	0	I	0.0	0.0	0:00.00	rcu_tasks_rude_kthread	1	0
14	root	20	0	0	0	0	I	0.0	0.0	0:00.00	rcu_tasks_trace_kthread	1	6
15	root	20	0	0	0	0	S	0.0	0.0	0:00.47	ksoftirqd/0	1	0
17	root	rt	0	0	0	0	S	0.0	0.0	0:00.08	migration/0	1	0
18	root	-51	0	0	0	0	S	0.0	0.0	0:00.00	idle_inject/0	1	0

گاهی اوقات می‌خواهیم که یک پردازش، برای مدیریت بهتر، بر روی یک core از پردازنده اجرا شود. با دستور taskset میتوانیم مشخص کنیم که پردازش روی کدام یک از core های پردازنده اجرا شود.

سوال امتیازی

دستور زیر را برای یک core دلخواه و یک دستور دلخواه اجرا کرده و خروجی آن را در خروجی دستور top در گزارش بیاورید.

پاسخ

دستور زیر را در ترمینال اجرا می‌کنیم:

`$ taskset --cpu-list 0 ls -l`

سپس با اجرای دستور top خروجی را مشاهده می‌کنیم:

```

top : 13:59:35 up 5:13, 2 users, load average: 0.95, 0.91, 0.89
Tasks: 342 total, 1 running, 341 sleeping, 0 stopped, 0 zombie
%Cpu(s): 4.2 us, 0.9 sy, 0.0 ni, 94.8 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 11801.6 total, 519.6 free, 5819.8 used, 6802.3 buff/cache
MiB Swap: 4096.0 total, 4096.0 free, 0.0 used, 5981.8 avail Mem

  PID USER      PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+  COMMAND
 3308 reza      20   0 5821420 422224 199836 S 37.7   3.5 58:43.35 gnome-shell
21826 reza     20   0 2789512 248492 108996 S 2.3   2.1 7:42.08 Isolated Web Co
4835 reza     20   0 12.8g 1.2g 249900 S 2.0 10.3 26:12.40 firefox
7080 reza     20   0 2653296 702156 236052 S 2.0   5.8 14:11.59 telegram-deskto
6074 reza     20   0 18.8g 244096 111168 S 1.7   2.0 5:32.15 Isolated Web Co
12222 reza     20   0 3093988 202548 105864 S 1.3   1.7 0:28.78 nautilus
28444 reza     20   0 717948 72364 55700 S 1.3   0.6 0:20.11 gnome-terminal-
3075 reza     9 -11 125504 23964 7636 S 1.0   0.2 0:24.24 pipewire-pulse
1466 root       0 -20   0      0      0 S 0.7   0.0 1:15.15 kworker/u25:1-i915_flip
3056 reza     9 -11 132968 15232 8536 S 0.7   0.1 0:20.94 pipewire
3067 reza     9 -11 406796 18728 13612 S 0.7   0.2 0:04.65 wireplumber
4027 reza     20   0 2566036 155704 96300 S 0.7   1.3 0:09.65 xdg-desktop-por
12868 reza     20   0 4620524 726348 170452 S 0.7   6.0 15:21.91 textstudio
13358 reza     20   0 2357532 135952 103344 S 0.7   1.1 2:42.01 nekoray
13368 reza     20   0 1255468 32180 19628 S 0.7   0.3 0:40.00 nekoray_core
30122 root       20   0      0      0 S 0.7   0.0 0:01.17 kworker/3:0-i915-unordered
 35 root       20   0      0      0 S 0.3   0.0 0:27.89 ksoftirqd/3
191 root      0 -20   0      0      0 S 0.3   0.0 0:02.04 kworker/6:1H-events_highpri
2044 root     20   0 1947272 43508 30976 S 0.3   0.4 0:18.93 containerd
3563 reza     20   0 388648 12444 7132 S 0.3   0.1 0:54.35 ibus-daemon
3929 reza     20   0 236900 7760 6952 S 0.3   0.1 0:17.31 ibus-engine-sim
3967 reza     20   0 1002952 10060 7944 S 0.3   0.1 0:00:27 gvfsd-trash
5174 reza     20   0 18.7g 157084 84472 S 0.3   1.3 4:32.08 WebExtensions
11802 reza     20   0 398300 10724 7728 S 0.3   0.1 0:01.06 gvfsd-recent
13258 reza     20   0 1238660 235128 57396 S 0.3   1.9 5:03.03 evince
22331 reza     20   0 2495776 142192 90116 S 0.3   1.2 0:05.24 Isolated Web Co
27771 root      20   0      0      0 S 0.3   0.0 0:00.44 kworker/u24:37-kcryptd/252:0
30270 reza     20   0 15728 6400 4096 R 0.3   0.1 0:00.12 top
  1 root      20   0 171492 15476 9144 S 0.0   0.1 0:07.29 systemd
  2 root      20   0      0      0 S 0.0   0.0 0:00.01 kthread
  3 root      0 -20   0      0      0 S 0.0   0.0 0:00.00 rcu_gp
  4 root      0 -20   0      0      0 S 0.0   0.0 0:00.00 rcu_par_gp
  5 root      0 -20   0      0      0 S 0.0   0.0 0:00.00 slub_flushwq
  6 root      0 -20   0      0      0 S 0.0   0.0 0:00.00 netns
  8 root      0 -20   0      0      0 S 0.0   0.0 0:00.00 kworker/8:0H-events_highpri
 11 root      0 -20   0      0      0 S 0.0   0.0 0:00.00 mm_percpu_wq
 12 root     20   0      0      0 S 0.0   0.0 0:00.00 rcu_tasks_kthread
 13 root     20   0      0      0 S 0.0   0.0 0:00.00 rcu_tasks_rude_kthread

namespaces for pid 3308, gnome-shell
cgroup: 4026531835, ipc: 4026531839, mnt: 4026531841, net: 4026531840, pid: 4026531836, time: 4026531834, user: 4026531837, uts: 4026531838

```