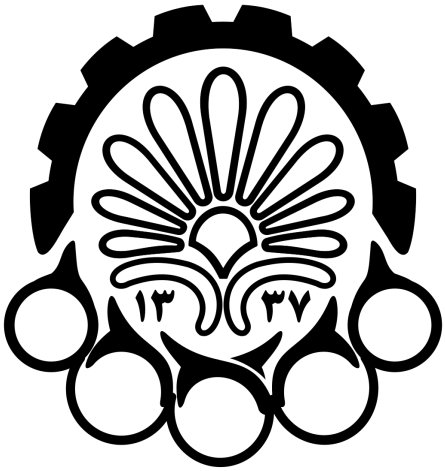
سیستم‌های عامل

دکتر جوادی



دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

رضا آدینه پور ۴۰۲۱۳۱۰۵۵

تمرین سری دوم

۲۴ آبان ۱۴۰۲

سیستم‌های عامل

تمرین سری دوم

رضا آدینه پور ۴۰۲۱۳۱۰۵۵

دانشکده مهندسی کامپیوتر

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 می‌شوند و بدون اینکه هسته از عملیات آن ها مطلع باشد کار می‌کنند

۴. زمان `context switch` در رابطه با نخ‌های سطح هسته کمتر طول می‌کشد.

۵. نخ‌های سطح کاربر نمی‌توانند به صورت موازی واقعی `true parallelism` در سیستم‌های چند هسته‌ای اجرا شوند زیرا توسط یک رشته در سطح هسته مدیریت می‌شوند.

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 کوتاهتری 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_valu
 pthread_exit(NULL);

 return 0;
}
```

پاسخ

این برنامه، یک برنامه چندنخی‌ست 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 ان 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 رشته ها 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 می توانند 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 می تونداشته 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()
```

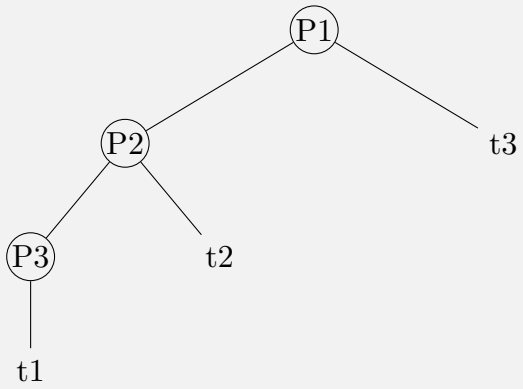
پاسخ

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 پردازه ها بعد از اتمام برنامه:
در ابتدا، یک پردازه وجود دارد. با اجرای دستور ()fork، یک پردازه جدید ایجاد می‌شود و تعداد پردازه‌ها به ۲ افزایش می‌یابد. در تابع func2، یک ()fork دیگر وجود دارد که تعداد پردازه‌ها به ۴ افزایش می‌یابد. پس از اتمام تمامی thread ها، تعداد پردازه‌ها 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 پردازه ها و مشخص کردن تعداد ترد های هرکدام:
پردازه اصلی P دارای ۳ ترد است: t1، t2، t3. پردازه اولیه که از طریق fork ایجاد شد دارای ۱ ترد است: t2. پردازه‌های دوم و سوم که از طریق fork در تابع func1 ایجاد شدند هر کدام دارای ۱ ترد است: t1 و t3.



۵. خالصه از فرآیند را شرح دهید:

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

سوال چهارم

با فرض موفقیت آمیز بودن اجرای دستورات `fork` و `execv` خروجی‌قطعه‌کد زیر را به صورت دقیق و با ذکر دلیل بیان‌کنید.

Listing 3: Some Code

```
int main()
{
  pid_t pid;
  pid = fork();
  if(pid == 0)
  {
        printf("process 1\n");
        char* args[] = {"ls", "-1", 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 برنامهٔ "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 پردازهٔ 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 پردازهٔ فرزند است. در این حالت:

- process 1 چاپ می‌شود.
- یک آرایه از رشته‌ها به نام args تعریف می‌شود که مسیر برنامهٔ "ls" و پارامترهای آن را مشخص می‌کند.
- تابع execv فراخوانی می‌شود تا برنامهٔ "ls" را با استفاده از آرگومان‌های مشخص شده اجرا کند. اگر این تابع با موفقیت اجرا شود، کنترل برنامه به برنامهٔ "ls" منتقل می‌شود و دستورات بعدی در کد اجرا نمی‌شوند.
- در صورتی که تابع execv با خطا مواجه شود و اجرای برنامهٔ "ls" انجام نشود، پیام process 1 finished چاپ می‌شود. این پیام هیچگاه نمایش داده نمی‌شود زیرا کنترل برنامه به برنامهٔ "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 پردازهٔ 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 برنامهٔ فوق، به ترتیب چاپ شدن پیام‌در این کد، در صورت موفقیت آمیز بودن تابع fork، یک پردازهٔ فرزند ایجاد می‌شود و بازگشتی غیر صفر دارد. اگر pid برابر ۰ باشد، به این معنی است که کد در حال اجرا در پردازهٔ فرزند است. در این حالت، Process 1 چاپ می‌شود و سپس تابع execv فراخوانی می‌شود تا برنامهٔ "ls" را با استفاده از آرگومان‌های مشخص شده اجرا کند. اگر این تابع با موفقیت اجرا شود، کنترل برنامه به برنامهٔ "ls" منتقل می‌شود و دستورات بعدی در کد اجرا نمی‌شوند. اگر تابع execv با خطا مواجه شود و اجرای برنامهٔ "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 پردازهٔ 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 برنامهٔ "ls" به عنوان نتیجهٔ اجرای execv وابسته به محتوای فایل‌ها و دایرکتوری‌های حاضر در مسیر /bin/ls است.

سوال پنجم

با فرض آنکه پردازه‌های producer و consumer به نحو زیر پیاده‌سازی شده اند. اگر سایز buffer برابر با ۵ باشد، و متغیر های 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 یک‌بار اجرای بدنه، حلقه producer یک‌بار بدنه حلقه consumer اجرا شود.
توضیحات: اگر تنها یک بار حلقهٔ بدنهٔ مصرف‌کننده و یک بار حلقهٔ بدنه تولیدکننده اجرا شونداگر تنها یک بار حلقهٔ 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.

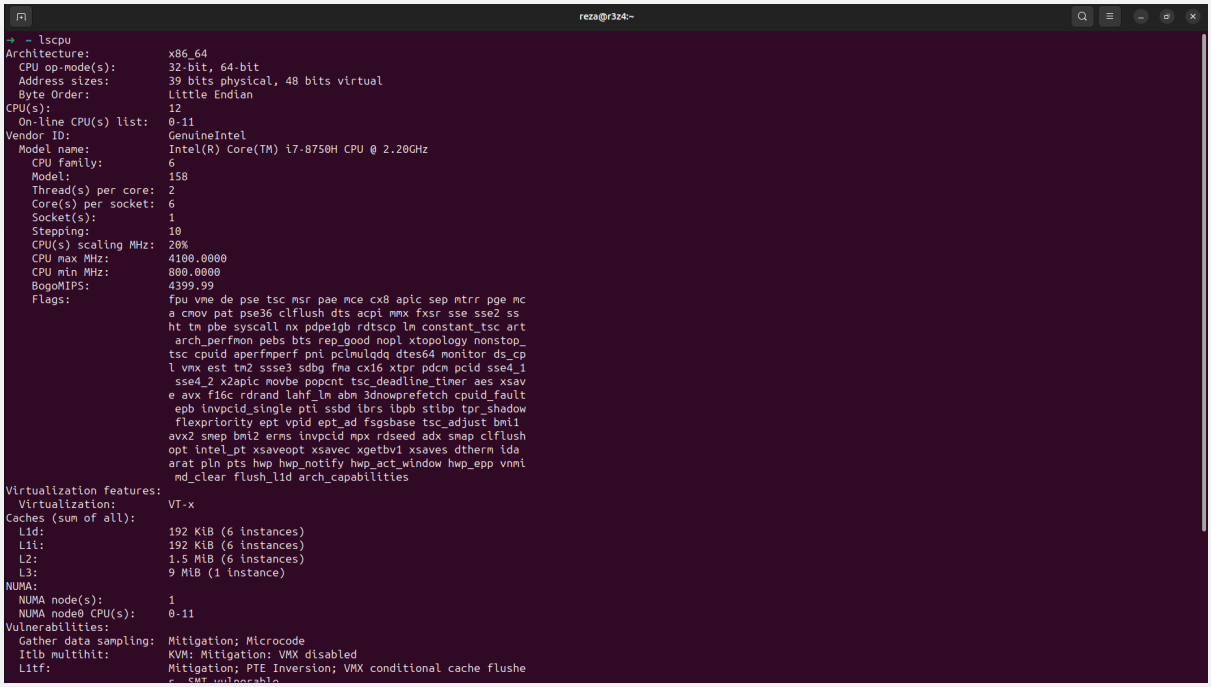
۲. به ازای هر دوبار اجرای بدنه، حلقه producer یک‌بار بدنه حلقه consumer اجرا شود.
توضیحات: اگر دو بار حلقه بدنه مصرف‌کننده و یک بار حلقه بدنه تولیدکننده اجرا شوند در این حالت، مقدار sum برابر با ۵ خواهد بود. تولیدکننده ۱۰ عدد از ۱ تا ۱۰ را در بافر قرار می‌دهد و مصرف‌کننده در دو بار اجرا، پنج عدد از بافر 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 سه‌بار اجرای بدنه، حلقه producer یک‌بار 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 می گذارد. خروجی دستور lscpu را در گزارش خود قرار دهید.

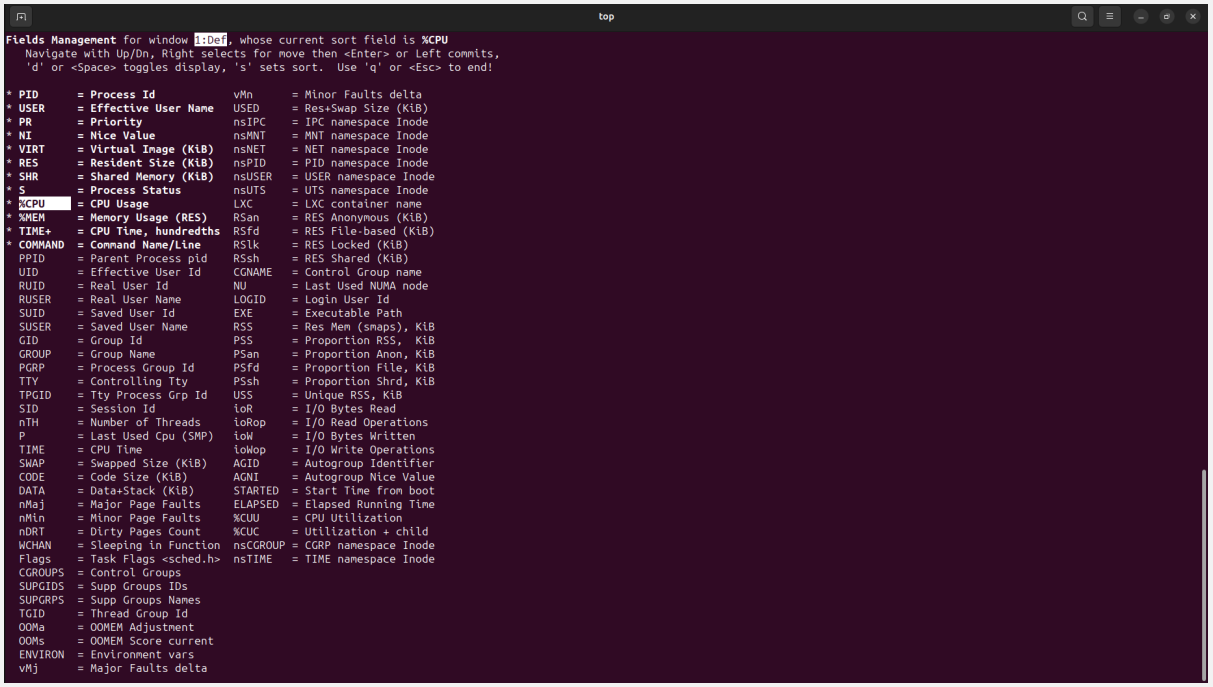
پاسخ



دستور top یکی از دستورات کاربردی در مدیریت پردازه 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.

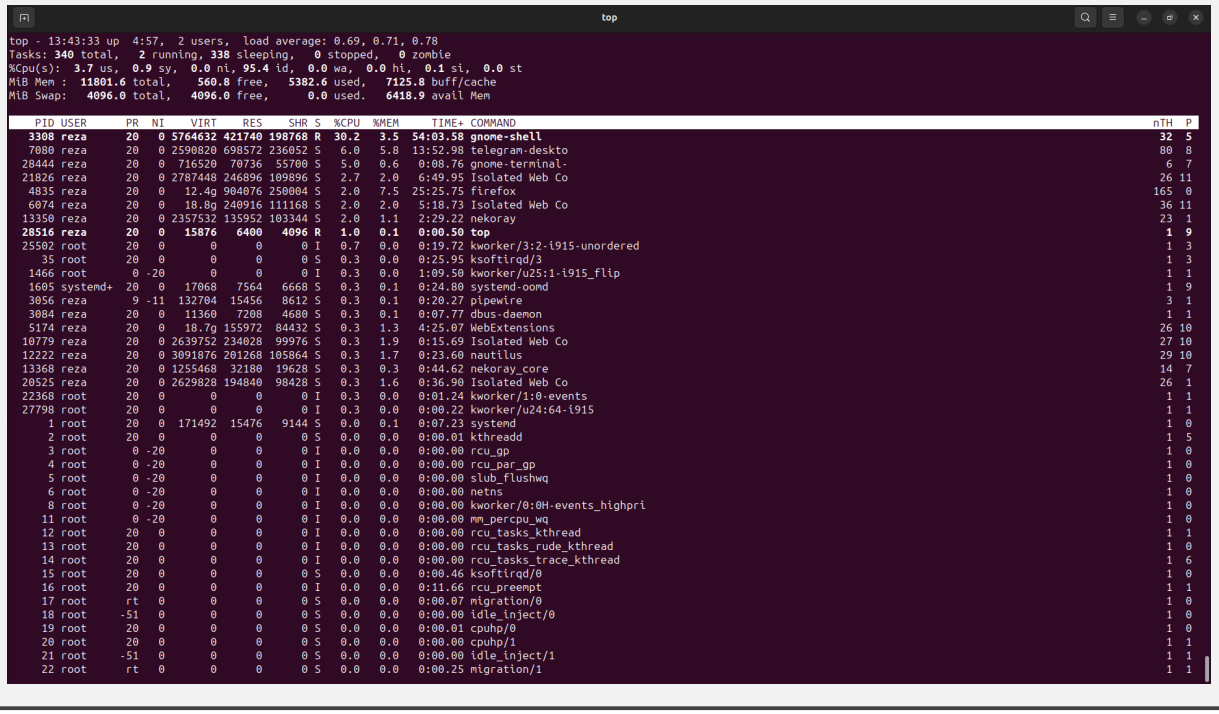
این دستور به صورت پیش فرض همه اطلاعات درمورد پردازه ها را به ما نمایش نمی دهد اما می‌توانیم به صورت دستی خودمان یک سری از اطلاعاتی که نیاز داریم را به ستون های اطلاعاتش اضافه و کم کنیم. برای اینکار پس از اجرای top دکمه f را فشار دهید. صفحه ای مشابه تصویر زیر برایتان نمایش داده میشود. با کلیدهای بالا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

پاسخ



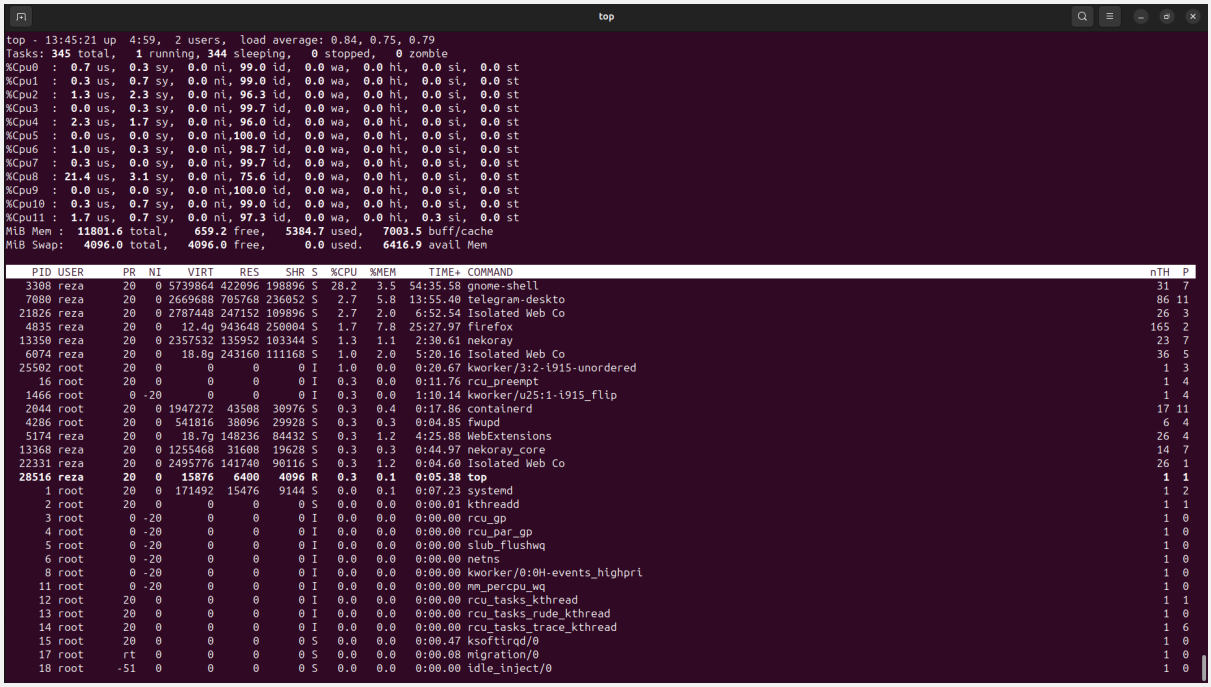
nTH تعداد thread های یک پردازه و p شماره آخرین هسته پردازنده ای که برای پردازه استفاده شده است را نمایش می دهد. این دو گزینه را انتخاب کنید و خروجی آن را در گزارشتان بیاورید.

پاسخ



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

پاسخ



گاهی اوقات می خواهیم که یک پردازه، برای مدیریت بهتر، بر روی یک core از پردازنده اجرا شود. با دستور taskset میتوانیم مشخص کنیم که پردازه 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: