آزمایشگاه سیستم‌عامل

جلسه ششم: برنامه‌نویسی چند فرآیندی و رسم نمودار توزیع نرمال

مدرس :مینا یوسف‌نژاد

نمونه‌گیری Sampling در تحقیقات علمی و آمار

در تحقیقات علمی و آمار، برای بررسی ویژگی‌های یک جامعه‌ی آماری (مثل گروهی از افراد، محصولات، یا پدیده‌ها)، می‌توان به جای مطالعه تمامی اعضای جامعه، بخشی از آن را به عنوان «نمونه» انتخاب کرد. به این فرآیند «نمونه‌گیری» می‌گویند. اهمیت نمونه‌گیری به چند دلیل است:

- صرفه‌جویی در زمان و هزینه: به‌جای جمع‌آوری داده‌ها از تمام اعضای جامعه، که ممکن است زمان‌بر و پرهزینه باشد، می‌توان با استفاده از نمونه‌گیری، اطلاعات کافی برای تخمین ویژگی‌های جامعه به دست آورد.
- عدم امکان دسترسی به همه اعضا: در بسیاری از مواقع، دسترسی به تمام اعضای جامعه ممکن نیست (برای مثال، وقتی جامعه‌ی آماری بسیار بزرگ یا حتی نامحدود است).

نمونه‌گیری در مقابل سرشماری قرار دارد. سرشماری به فرآیندی اشاره دارد که تمام اعضای جامعه مورد بررسی قرار می‌گیرند. اگرچه سرشماری نتایج دقیقی ارائه می‌دهد، اما اغلب به دلیل محدودیت‌های زمانی و مالی، نمی‌تواند در همه شرایط انجام شود.

نمونه‌گیری برای محاسبه‌ی عدد π

یکی از روش‌های جالب استفاده از نمونه‌گیری در محاسبات ریاضی، تخمین عدد π است. در این روش، با تولید زوج عددهای تصادفی و بررسی تعداد دفعاتی که این زوج‌ها درون یک دایره (نسبت به یک مربع مرجع) قرار می‌گیرند، می‌توان عدد پی را تخمین زد.

مراحل تخمین عدد π:

- تعداد زیادی زوج عدد تصادفی تولید کنید.
- بررسی کنید که هر زوج در داخل دایره‌ای با شعاع مشخص قرار دارد یا خیر.
- نسبت تعداد زوج‌های داخل دایره به کل زوج‌های تولید شده تقریباً برابر با عدد π خواهد بود.

شبیه‌سازی یک سری محاسبات تصادفی و سپس تحلیل نتایج آنها به‌صورت سری و موازی

- **هدف اصلی:** برنامه تعداد زیادی نمونه تصادفی تولید می‌کند و سپس این داده‌ها را تجزیه و تحلیل می‌کند تا یک هیستوگرام از توزیع آنها بسازد.
- **شبیه‌سازی آزمایش‌های تصادفی:** در این برنامه، برای هر نمونه، یک سری ۱۲ عدد تصادفی تولید می‌شود که هر کدام بین ۰ تا ۹۹ هستند. سپس این اعداد در یک عملیات خاص که به‌صورت افزایش یا کاهش شمارنده **counter** عمل می‌کند، استفاده می‌شوند.
  - اگر عدد تصادفی بزرگتر از ۴۹ باشد، شمارنده افزایش می‌یابد.
  - اگر عدد تصادفی کمتر از ۴۹ باشد، شمارنده کاهش می‌یابد.

پس از انجام این محاسبات برای ۱۲ عدد، مقدار نهایی شمارنده (که می‌تواند مثبت یا منفی باشد) در یک آرایه به نام هیستوگرام ذخیره می‌شود.

- **هدف از هیستوگرام:** هیستوگرام یک نمودار است که نشان می‌دهد هر مقدار شمارنده (از -۱۲ تا +۱۲) چند بار در کل آزمایش‌ها تکرار شده است. به عبارت دیگر، هر فرآیند تصادفی می‌تواند یک نتیجه متفاوت تولید کند، و هیستوگرام تعداد دفعاتی که هر نتیجه رخ داده است را نشان می‌دهد.

پیاده‌سازی برنامه در حالت سریال

در این مرحله، کدی می‌نویسیم که بر اساس تعریف مسئله، توزیع نرمال را با استفاده از نمونه‌گیری به صورت سریال پیاده‌سازی می‌کند. کد به این صورت عمل می‌کند که:

- آرایه‌ای از نتایج نمونه‌گیری با نام hist ایجاد می‌کند که ۲۵ خانه دارد.
- برای هر نمونه، یک متغیر counter مقداردهی اولیه می‌شود.
- در ۱۲ مرحله، برای هر نمونه‌گیری یک عدد تصادفی تولید می‌شود. اگر این عدد بزرگتر یا مساوی ۴۹ باشد، counter افزایش می‌یابد؛ در غیر این صورت، کاهش پیدا می‌کند.
- مقدار counter در هر مرحله در آرایه‌ی hist ثبت می‌شود.

سپس زمان اجرای برنامه برای تعداد نمونه‌های مختلف (۵۰۰۰، ۵۰۰۰۰، و ۵۰۰۰۰۰) اندازه‌گیری می‌شود و در جدول وارد می‌گردد.

تولید عدد تصادفی

در بسیاری از زبان‌های برنامه‌نویسی، برای تولید اعداد تصادفی از یک الگوریتم خاص استفاده می‌شود که معمولاً از یک عدد ثابت (که به آن seedگفته می‌شود) برای شروع تولید اعداد تصادفی استفاده می‌کند. این عدد باید هر بار که برنامه اجرا می‌شود متفاوت باشد تا از تولید همان دنباله از اعداد تصادفی جلوگیری شود.

srand(): تابعی است که برای تنظیم seedاستفاده می‌شود. این تابع به همراه تابع rand() که اعداد تصادفی تولید می‌کند کار می‌کند.

time(NULL): این تابع زمان جاری سیستم را به‌صورت تعداد ثانیه‌ها از زمان ۱ ژانویه ۱۹۷۰ (زمان UNIX) باز می‌گرداند. زمانی که این دستور اجرا می‌شود، زمان جاری را به‌عنوان seed برای srand() استفاده می‌کند.

در نتیجه، با استفاده از time(NULL)، هر بار که برنامه اجرا می‌شود، seedمتفاوتی برای تولید اعداد تصادفی تعیین می‌شود (چون زمان هر بار متفاوت است). این امر موجب می‌شود که دنباله اعداد تصادفی که تولید می‌شود، در هر اجرای برنامه متفاوت باشد.

```
srand(time(NULL));
```

رسم هیستوگرام

```c
void printHistogram(int* hist) {
    int i, j;
    for (i = 0; i < 25; i++) {
        printf("hist[%d]: ", i - 12);
        for (j = 0; j < hist[i]; j++) {
            printf("*");
        }
        printf("\n");
    }
}
```

```c
void saveHistogramToFile(int* hist, int size) {
    FILE *file = fopen("histogram_data.txt", "w");
    for (int i = 0; i < size; i++) {
        fprintf(file, "%d %d\n", i - 12, hist[i]);
    }
    fclose(file);
}
```

real: زمان کلی اجرا از لحظه شروع تا پایان برنامه است، و شامل تمام تأخیرها، از جمله زمان‌های انتظار برای منابع سیستم یا سایر برنامه‌های در حال اجرا است.به این زمان Elapsed Time یا زمان واقعی نیز گفته می‌شود. معمولاً این زمان بیشترین مقدار را دارد، زیرا شامل همه زمان‌هایی است که برنامه در حال اجرا بوده است.

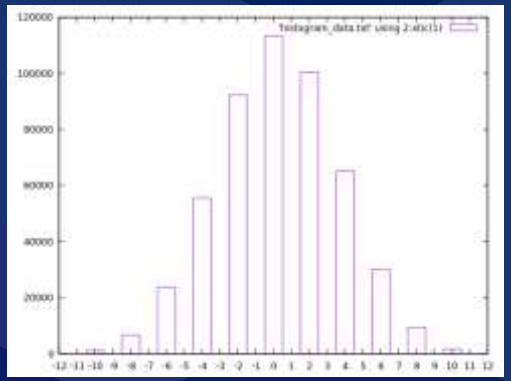
user: این زمان نشان‌دهنده مقدار زمانی است که CPU صرف اجرای کدهای کاربر(User Mode) کرده است. این زمان شامل پردازش‌هایی است که در سطح برنامه شما انجام شده و شامل توابع سیستم‌عامل نمی‌شود.

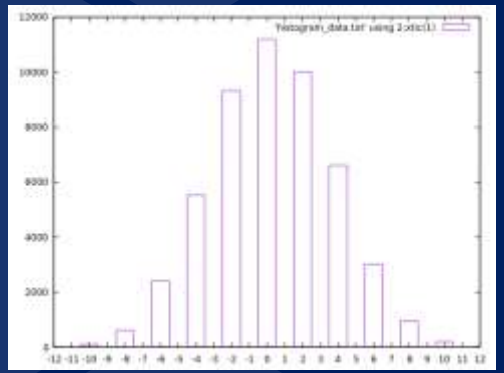
sys: این مقدار نشان‌دهنده زمانی است که CPUدر سطح کرنل (Kernel Mode) صرف اجرای توابع سیستم‌عامل کرده است، مانند دسترسی به فایل‌ها، خواندن و نوشتن از طریق شبکه یا ارتباطات بین‌پردازشی.

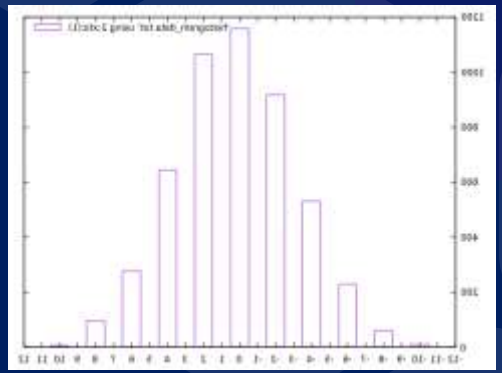
```
real    0m0.033s
user    0m0.003s
sys     0m0.000s
```

SingleProsecc

500000	50000	5000	تعداد نمونه
0m0.078s	0m0.011s	0m0.004s	زمان اجرا







استفاده از فرآیندهای موازی

برنامه به جای اینکه همه محاسبات را در یک فرآیند انجام دهد، از چندین فرآیند موازی استفاده می‌کند تا هر فرآیند بخشی از محاسبات را انجام دهد. این امر موجب افزایش سرعت برنامه در حالت‌هایی می‌شود که تعداد نمونه‌های زیادی وجود دارد.

- ابتدا تعداد کل نمونه‌ها (SAMPLE_COUNT) را به تعداد فرآیندها تقسیم می‌کنیم.
- سپس برای هر فرآیند، محاسبه بخشی از این نمونه‌ها انجام می‌شود.
- در نهایت، فرآیندهای فرزند کار خود را انجام داده و سپس والد منتظر می‌ماند تا همه آن‌ها تمام شوند.

تولید عدد تصادفی

```
srand(time(NULL) ^ (getpid()));
```

این دستور مشابه دستور قبلی است اما با یک تفاوت مهم: علاوه بر time(NULL)که زمان جاری را به‌عنوان seedمی‌گیرد، از getpid() نیز استفاده می‌شود.

getpid():این تابع شناسه فرآیند جاری Process IDیا PID را برمی‌گرداند. هر فرآیند در سیستم‌عامل یک شناسه منحصر به فرد دارد که به آن PID می‌گویند. بنابراین، اگر چندین فرآیند همزمان اجرا شوند، PID آن‌ها متفاوت خواهد بود.

عملگر ^ (XOR): این عملگر منطقی به‌صورت بیتی دو مقدار را با هم مقایسه کرده و در صورت تفاوت در هر بیت، آن‌را به ۱ تبدیل می‌کند. به‌عبارت دیگر، این عملگر دو عدد را از نظر بیتی ترکیب می‌کند.

time(NULL) ^ getpid(): ترکیب زمان جاری و شناسه فرآیند جاری به‌وسیله عملگر XOR انجام می‌شود.

این ترکیب باعث می‌شود که seed تولید اعداد تصادفی نه تنها بر اساس زمان سیستم بلکه بر اساس شناسه فرآیند نیز متفاوت باشد. به‌این‌ترتیب، حتی اگر دو فرآیند در زمان مشابه شروع به اجرا کنند، با استفاده از getpid() یک seedمختلف به‌دست می‌آید.

مثال

فرض کنید که دو فرآیند فرزند به‌طور همزمان از یک برنامه استفاده می‌کنند:

در فرآیند اول: time(NULL) برابر با ۱۷۰۰۰۰۰۰۰۰ و getpid() برابر با ۱۲۳۴۵ باشد.

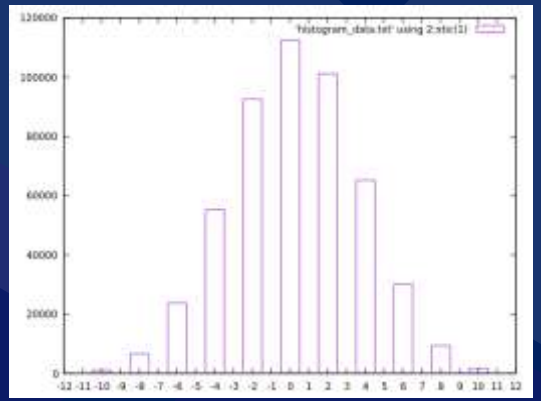
بنابراین seed برای این فرآیند ۱۷۰۰۰۰۰۰۰۰ ^ ۱۲۳۴۵ خواهد بود.

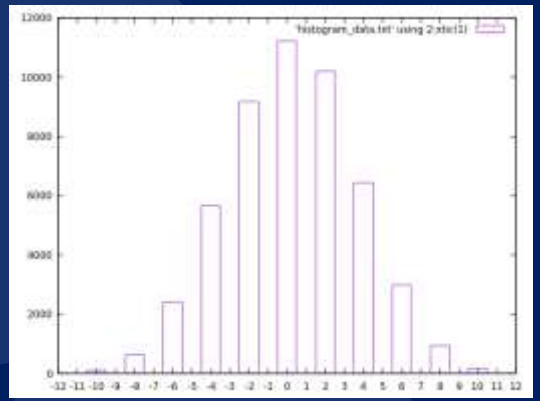
در فرآیند دوم: time(NULL) همان مقدار ۱۷۰۰۰۰۰۰۰۰ و getpid() برابر با ۱۲۳۴۶ باشد.

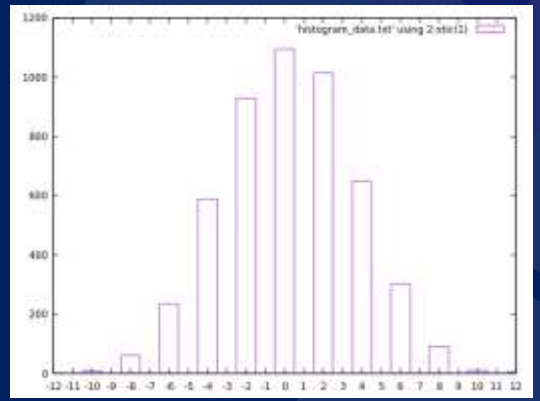
بنابراین seed برای این فرآیند ۱۷۰۰۰۰۰۰۰۰ ^ ۱۲۳۴۶ خواهد بود.

در اینجا با اینکه time(NULL) مشابه است (چون فرآیندها تقریباً همزمان اجرا می‌شوند)، استفاده از getpid() باعث می‌شود که دنباله تصادفی هر فرآیند متفاوت باشد.

500000	50000	5000	تعداد نمونه
0m0.024s	0m0.005s	0m0.003s	زمان اجرا







SingleProsecc

500000	50000	5000	تعداد نمونه
0m0.078s	0m0.011s	0m0.004s	زمان اجرا

MultiProsecc

500000	50000	5000	تعداد نمونه
0m0.024s	0m0.005s	0m0.003s	زمان اجرا

آیا این برنامه درگیر شرایط مسابقه Race Condition می‌شود؟

بله، برنامه چندفرآیندی درگیر شرایط مسابقه می‌شود. این مشکل زمانی رخ می‌دهد که فرآیندهای فرزند به‌طور همزمان به یک منبع مشترک، یعنی آرایه‌ی hist دسترسی پیدا می‌کنند. در نتیجه، هر فرآیند می‌تواند در همان لحظه‌ای که فرآیند دیگری در حال نوشتن در آرایه است، مقدار جدیدی را بنویسد، که باعث می‌شود نتایج نهایی نادرست یا پیش‌بینی‌ناپذیر باشند.

راه‌حل برای شرایط مسابقه: برای حل این مشکل، می‌توان از تکنیک‌های همگام‌سازی فرآیندها استفاده کرد.

استفاده از Shared Memory با استفاده از حافظه‌ی اشتراکی، می‌توان یک منطقه‌ی اشتراکی برای ذخیره‌ی آرایه‌ی hist تعریف کرد.

تجمع نتایج در حافظه محلی هر فرآیند و ادغام نتایج در انتها: به هر فرآیند فرزند یک نسخه محلی از آرایه‌ی hist اختصاص دهید.پس از پایان محاسبات هر فرآیند، نتایج محلی را به فرآیند والد بازگردانید و در آنجا نتایج نهایی را با هم جمع کنید.