Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Отчёт
по лабораторной работе №1

по дисциплине «Операционные системы»

Вариант: *vfork, ema-search-str, sort*

Выполнил:
Рудкевич И. А., группа P3306

Преподаватель:
Тюрин И. Н.

Санкт-Петербург
~ 2024 ~

Оглавление

Задание

Часть 1. Запуск программ

Необходимо реализовать собственную оболочку командной строки - shell. Выбор ОС для реализации производится на усмотрение студента. Shell должен предоставлять пользователю возможность запускать программы на компьютере с переданными аргументами командной строки и после завершения программы показывать реальное время ее работы (подсчитать самостоятельно как «время завершения» – «время запуска»).

Часть 2. Мониторинг и профилирование

Разработать комплекс программ-нагрузчиков по варианту, заданному преподавателем. Каждый нагрузчик должен, как минимум, принимать параметр, который определяет количество повторений для алгоритма, указанного в задании. Программы должны нагружать вычислительную систему, дисковую подсистему или обе подсистемы сразу. Необходимо скомпилировать их без опций оптимизации компилятора.
Перед запуском нагрузчика, попробуйте оценить время работы вашей программы или ее результаты (если по варианту вам досталось измерение чего либо). Постарайтесь обосновать свои предположения. Предположение можно сделать, основываясь на свой опыт, знания ОС и характеристики используемого аппаратного обеспечения.

1. Запустите программу-нагрузчик и зафиксируйте метрики ее работы с помощью инструментов для профилирования. Сравните полученные результаты с ожидаемыми. Постарайтесь найти объяснение наблюдаемому.
2. Определите количество нагрузчиков, которое эффективно нагружает все ядра процессора на вашей системе. Как распределяются времена USER%, SYS%, WAIT%, а также реальное время выполнения нагрузчика, какое количество переключений контекста (вынужденных и невынужденных) происходит при этом?
3. Увеличьте количество нагрузчиков вдвое, втрое, вчетверо. Как изменились времена, указанные на предыдущем шаге? Как ведет себя ваша система?
4. Объедините программы-нагрузчики в одну, реализованную при помощи потоков выполнения, чтобы один нагрузчик эффективно нагружал все ядра вашей системы. Как изменились времена для того же объема вычислений? Запустите одну, две, три таких программы.
5. Добавьте опции агрессивной оптимизации для компилятора. Как изменились времена? На сколько сократилось реальное время исполнения программы нагрузчика?

Ограничения

Программа (комплекс программ) должна быть реализован на языке C, C++.
Дочерние процессы должны быть созданы через заданные системные вызовы выбранной операционной системы, с обеспечением корректного запуска и и завершения процессов. Запрещено использовать высокоуровневые абстракции над системными вызовами. Необходимо использовать, в случае Unix, процедуры libc.

Листинг исходного кода

shell.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <unistd.h>

#define MAX_INPUT_SIZE 128
#define MAX_ARGS 16

void execute_command(char *input) {
  pid_t pid;
  int status;
  char *args[MAX_ARGS];
  char *token;
  time_t start_time, end_time;

  token = strtok(input, " \n");
  int i = 0;
  while (token != NULL && i < MAX_ARGS - 1) {
    args[i++] = token;
    token = strtok(NULL, " \n");
  }
  args[i] = NULL;

  start_time = time(NULL);
  pid = vfork();
  if (pid < 0) {
    perror("vfork failed");
    exit(1);
  } else if (pid == 0) {
    execvp(args[0], args);
    perror("exec failed");
    _exit(1);
  } else {
    waitpid(pid, &status, 0);
    end_time = time(NULL);
    if (WIFEXITED(status)) {
      printf("Program finished with exit status %d\n", WEXITSTATUS(status));
    }
    printf("Execution time: %lf seconds\n", (double) (end_time - start_time) /
CLOCKS_PER_SEC);
  }
}

int main(void) {
  char input[MAX_INPUT_SIZE];

  while (1) {
    printf("cumass> ");
    if (fgets(input, MAX_INPUT_SIZE, stdin) == NULL) {
      break;
    }

    if (strlen(input) > 1) {
      execute_command(input);
    }
  }
  return 0;
}
```

sort.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// generate array with random values
void generate_array(int *arr, int size) {
  for (int i = 0; i < size; i++) {
    arr[i] = rand() % 1000;
  }
}

void quicksort(int *arr, int low, int high) {
  if (low < high) {
    int mid = arr[high];
    int i = low - 1;

    for (int j = low; j < high; j++) {
      if (arr[j] < mid) {
        i++;
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
      }
    }
    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;

    quicksort(arr, low, i);
    quicksort(arr, i + 2, high);
  }
}

void bubble_sort(int *arr, int size) {
  for (int i = 0; i < size - 1; i++) {
    for (int j = 0; j < size - i - 1; j++) {
      if (arr[j] > arr[j + 1]) {
        int temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
      }
    }
  }
}

int main(int argc, char *argv[]) {
  if (argc != 3) {
    fprintf(stderr, "Usage: %s <arr_length> <repeats>\n", argv[0]);
    return 1;
  }

  int n = atoi(argv[1]);
  int repeats = atoi(argv[2]);
  int *arr = malloc(n * sizeof(int));

  clock_t start_total = clock();
  clock_t start;
  clock_t end;

  for (int r = 0; r < repeats; r++) {
    start = clock();
    generate_array(arr, n);
    quicksort(arr, 0, n - 1);
    end = clock();
    printf("[%d] Quicksort execution time: %lf seconds\n", r+1, (double) (end -
start) / CLOCKS_PER_SEC);
  }
```

```
  clock_t end_total = clock();

  free(arr);
  printf("\n");
  printf(">>> Total execution time: %lf seconds <<<\n\n", (double) (end_total -
start_total) / CLOCKS_PER_SEC);

  return 0;
}
```

ema_search.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define BUFFER_SIZE 1024

void find_substring(const char *buffer, const char *substring, int line_number) {
  const char *ptr = buffer;
  int position;

  while ((ptr = strstr(ptr, substring)) != NULL) {
    position = ptr - buffer;
    printf("Found substring at line %d, position %d\n", line_number, position);
    ptr++;
  }
}

int process_file(const char *filename, const char *substring) {
  FILE *file = fopen(filename, "r");
  if (file == NULL) {
    perror("Error opening file");
    return EXIT_FAILURE;
  }

  char buffer[BUFFER_SIZE];
  int line_number = 0;
  while (fgets(buffer, BUFFER_SIZE, file)) {
    line_number++;
    find_substring(buffer, substring, line_number);
  }
  fclose(file);

  return EXIT_SUCCESS;
}

int main(int argc, char *argv[]) {
  if (argc < 4) {
    fprintf(stderr, "Usage: %s <filename> <substring> <repeats>\n", argv[0]);
    return EXIT_FAILURE;
  }

  const char *filename = argv[1];
  const char *substring = argv[2];
  size_t repeats = atoi(argv[3]);

  if (strlen(substring) > BUFFER_SIZE) {
    fprintf(stderr, "Substring is larger than buffer size");
  }

  clock_t total_start = clock();
  for (int r = 0; r < repeats; r++) {
    clock_t start = clock();
    if (process_file(filename, substring) != 0) {
      fprintf(stderr, "[%d] AHTUNG AHTUNG SOME ERROR OCCURRED\n", r);
    }
```

```
    printf("Execution time: %lf seconds\n\n", (double) (clock() - start) /
CLOCKS_PER_SEC);
  }
  clock_t total_end = clock();

  printf("\n");
  printf(">>> Total execution time: %lf seconds <<<\n\n", (double) (total_end -
total_start) / CLOCKS_PER_SEC);

  return EXIT_SUCCESS;
}
```

parallel.sh

```
#!/bin/bash

# Set the number of parallel runs
N=$1

if [ -z "$N" ] || [ -z "$2" ]; then
  echo "Usage: $0 <parallel_runs> <program> [args...]"
  exit 1
fi

PROGRAM=$2
shift 2

# Handle Ctrl+C and terminate all running processes
cleanup() {
  echo "Interrupt received. Terminating all processes..."
  kill $(jobs -p) 2>/dev/null
  exit 1
}

trap cleanup SIGINT

for ((i=1; i<=N; i++)); do
  $PROGRAM "$@" &
done

wait

echo "All $N instances have finished."
```

Измерения

Запустим программу-нагрузчик и зафиксируем метрики ее работы с помощью инструментов для профилирования

Sort

Программа-нагрузчик скомпилирована с флагом `-O0` *(без оптимизации).*

Запустим `sort` 100 раз для массива длиной 10000:

```
$ perf stat ./sort 10000 100
```

```
>>> Total execution time: 0.099012 seconds <<<


 Performance counter stats for './bin/sort 10000 100':

            99.54 msec task-clock                #    0.974 CPUs utilized
                1      context-switches          #   10.047 /sec
                0      cpu-migrations            #    0.000 /sec
               59      page-faults               #  592.749 /sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

       0.102210375 seconds time elapsed

       0.099134000 seconds user
       0.002002000 seconds sys
```

Теперь запустим `sort` 100 раз для массива длиной 1000000, чтобы успеть зафиксировать нагрузку процессора

Воспользуемся утилитой `top` с флагом `-p` для мониторинга нашей программы программы-нагрузчика.
Также воспользуемся командой `pgrep` с флагом `-f`, чтобы определить `pid` нашей программы-нагрузчика.
В итоге получим:

```
$ perf stat ./sort 1000000 100
```

```
top - 13:09:51 up 16:10,  0 users,  load average: 1.35, 0.73, 0.76
Tasks:   1 total,   1 running,   0 sleeping,   0 stopped,   0 zombie
%Cpu(s): 12.5 us,  0.0 sy,  0.0 ni, 87.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3923.1 total,   2122.3 free,    556.6 used,   1244.2 buff/cache
MiB Swap:   1024.0 total,   1024.0 free,      0.0 used.   3172.0 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  459 root      20   0    6096   5276   1280 R 100.0   0.1   1:25.62 sort
```

```
>>> Total execution time: 113.206205 seconds <<<


 Performance counter stats for './bin/sort 1000000 100':

         113211.93 msec task-clock                #    1.000 CPUs utilized
                 5      context-switches          #    0.044 /sec
                 0      cpu-migrations            #    0.000 /sec
               532      page-faults               #    4.699 /sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

     113.216532218 seconds time elapsed

     113.197506000 seconds user
       0.015044000 seconds sys
```

Как можно заметить, один процесс нагружает одно ядро на 100%.

Для увеличения количества нагрузчиков воспользуемся скриптом `parallel.sh`
Увеличим число процессов вдвое, получим:

```
$ bash ./parallel.sh 2 ./bin/sort 1000000 100
```

```
top - 14:45:17 up 17:35,  0 users,  load average: 1.23, 0.90, 0.77
Tasks:   7 total,   3 running,   4 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.7 us,  0.7 sy,  0.0 ni, 98.0 id,  0.0 wa,  0.0 hi,  0.7 si,  0.0 st
%Cpu1  :  0.3 us,  0.3 sy,  0.0 ni, 99.0 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu2  :  0.3 us,  1.0 sy,  0.0 ni, 98.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :  0.3 us,  0.3 sy,  0.0 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :  0.7 us,  0.7 sy,  0.0 ni, 98.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  :  0.3 us,  0.3 sy,  0.0 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3923.1 total,   2114.3 free,    563.6 used,   1245.3 buff/cache
MiB Swap:   1024.0 total,   1024.0 free,      0.0 used.   3164.3 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  632 root      20   0    6096   5340   1280 R 100.0   0.1   0:41.03 sort
  633 root      20   0    6096   5272   1280 R 100.0   0.1   0:41.03 sort
    1 root      20   0    4116   2944   2688 S   0.0   0.1   0:00.02 bash
    9 root      20   0    4636   3840   2816 S   0.0   0.1   0:01.32 bash
  501 root      20   0    4120   3200   2816 S   0.0   0.1   0:00.09 bash
  631 root      20   0    3852   2816   2560 S   0.0   0.1   0:00.00 bash
  634 root      20   0    6864   2944   2432 R   0.0   0.1   0:00.02 top
```

```
>>> Total execution time: 119.426033 seconds <<<
>>> Total execution time: 119.486605 seconds <<<
```

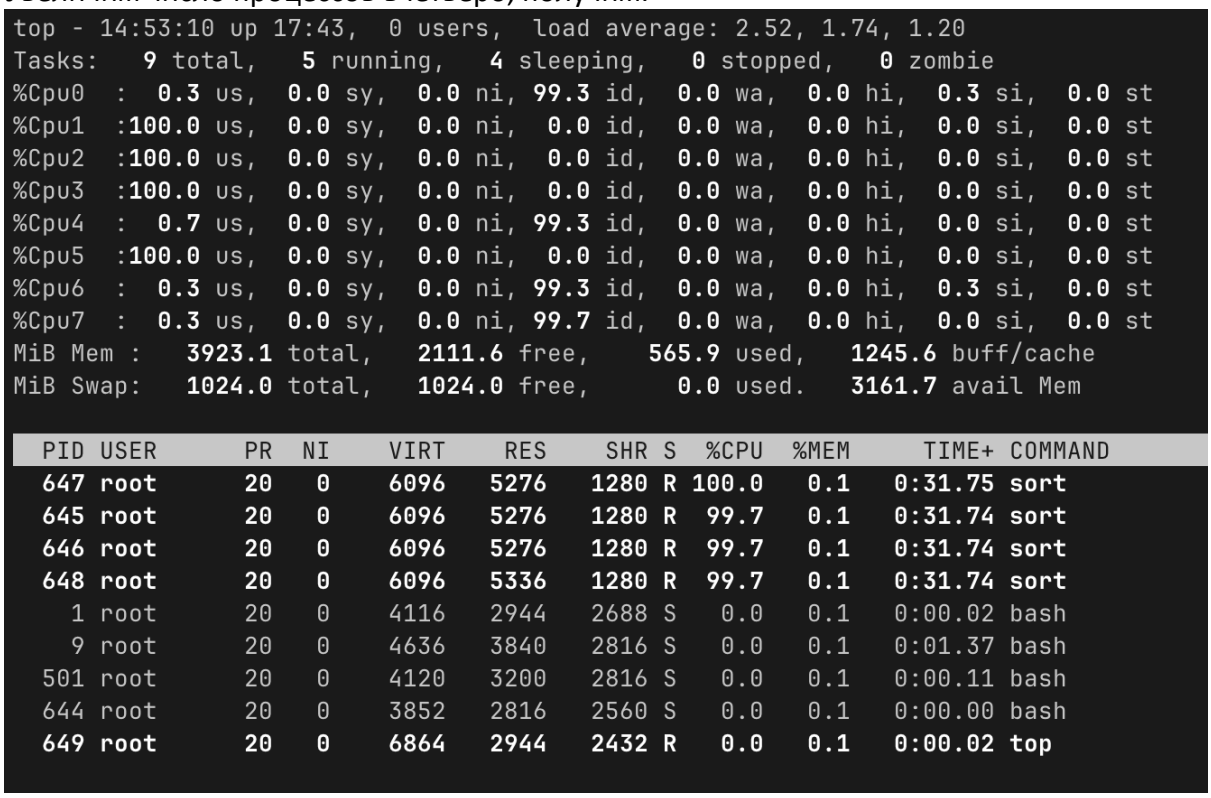
Увеличим число процессов втрое, получим:

```
$ bash ./parallel.sh 3 ./bin/sort 1000000 100
```

```
top - 14:49:52 up 17:40,  0 users,  load average: 1.96, 1.22, 0.94
Tasks:   8 total,   4 running,   4 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.0 us,  0.3 sy,  0.0 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu1  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :  0.3 us,  0.0 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3923.1 total,   2113.2 free,    564.5 used,   1245.4 buff/cache
MiB Swap:   1024.0 total,   1024.0 free,      0.0 used.   3163.2 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  640 root      20   0    6096   5336   1280 R 100.0   0.1   0:42.70 sort
  638 root      20   0    6096   5340   1280 R 100.0   0.1   0:42.70 sort
  639 root      20   0    6096   5276   1280 R 100.0   0.1   0:42.69 sort
    1 root      20   0    4116   2944   2688 S   0.0   0.1   0:00.02 bash
    9 root      20   0    4636   3840   2816 S   0.0   0.1   0:01.34 bash
  501 root      20   0    4120   3200   2816 S   0.0   0.1   0:00.11 bash
  637 root      20   0    3852   2816   2560 S   0.0   0.1   0:00.01 bash
  641 root      20   0    6864   2944   2432 R   0.0   0.1   0:00.01 top
```

```
>>> Total execution time: 123.410891 seconds <<<
>>> Total execution time: 123.458051 seconds <<<
>>> Total execution time: 123.460510 seconds <<<
```

Увеличим число процессов вчетверо, получим:




```
>>> Total execution time: 124.220246 seconds <<<
>>> Total execution time: 124.305672 seconds <<<
>>> Total execution time: 124.313848 seconds <<<
>>> Total execution time: 124.340852 seconds <<<
```

Для восьми процессов получим:

```
$ bash ./parallel.sh 8 ./bin/sort 1000000 100
```

```
top - 14:58:41 up 17:49,  0 users,  load average: 3.20, 1.96, 1.47
Tasks:  13 total,   9 running,   4 sleeping,   0 stopped,   0 zombie
%Cpu0  : 99.7 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu1  : 99.6 us,  0.4 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  : 99.7 us,  0.3 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3923.1 total,   2086.6 free,    590.7 used,   1245.8 buff/cache
MiB Swap:   1024.0 total,   1024.0 free,      0.0 used.   3136.6 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  655 root      20   0    6096   5276   1280 R 100.0   0.1   0:27.78 sort
  653 root      20   0    6096   5336   1280 R 100.0   0.1   0:27.80 sort
  654 root      20   0    6096   5340   1280 R 100.0   0.1   0:27.70 sort
  656 root      20   0    6096   5272   1280 R 100.0   0.1   0:27.80 sort
  658 root      20   0    6096   5248   1280 R 100.0   0.1   0:27.78 sort
  657 root      20   0    6096   5332   1280 R  99.7   0.1   0:27.72 sort
  659 root      20   0    6096   5268   1280 R  99.7   0.1   0:27.88 sort
  652 root      20   0    6096   5340   1280 R  98.0   0.1   0:27.72 sort
    1 root      20   0    4116   2944   2688 S   0.0   0.1   0:00.02 bash
    9 root      20   0    4636   3840   2816 S   0.0   0.1   0:01.39 bash
  501 root      20   0    4120   3200   2816 S   0.0   0.1   0:00.11 bash
  649 root      20   0    6864   2944   2432 R   0.0   0.1   0:00.56 top
  651 root      20   0    3852   2816   2560 S   0.0   0.1   0:00.01 bash
```

```
>>> Total execution time: 157.343920 seconds <<<
>>> Total execution time: 157.374270 seconds <<<
>>> Total execution time: 157.865858 seconds <<<
>>> Total execution time: 157.966960 seconds <<<
>>> Total execution time: 158.190366 seconds <<<
>>> Total execution time: 158.226615 seconds <<<
>>> Total execution time: 158.623097 seconds <<<
>>> Total execution time: 158.161338 seconds <<<
```

Ema Search String

Программа-нагрузчик скомпилирована с флагом `-O0` *(без оптимизации).*

Поиск будем осуществлять по файлу `out.txt`:

```
root@0a3dfb51906c:/workspace# ls -lh ./text/out.txt
-rw-r--r-- 1 root root 301M Nov  9 11:42 ./text/out.txt
```

Запустим `ema_search` 1 раз и будем искать подстроку `QalIVKumQfiPgyJrzwoQMmPlmtfUcM`:

```
$ perf stat ./ema_search ./text/out.txt
QalIVKumQfiPgyJrzwoQMmPlmtfUcM 1
```

```
Found substring at position 31933353
Execution time: 0.120481 seconds


>>> Total execution time: 0.120516 seconds <<<


 Performance counter stats for './bin/ema_search text/out.txt QalIVKumQfiPgyJrzwoQ 1':

            121.15 msec task-clock                #    0.847 CPUs utilized
               148      context-switches          #    1.222 K/sec
                 2      cpu-migrations            #   16.508 /sec
                50      page-faults               #  412.705 /sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

       0.143114250 seconds time elapsed

       0.096764000 seconds user
       0.025936000 seconds sys
```

Теперь запустим `ema_search` 1000 раз, чтобы успеть зафиксировать нагрузку процессора

Воспользуемся утилитой `top` с флагом `-p` для мониторинга нашей программы программы-нагрузчика.
Также воспользуемся командой `pgrep` с флагом `-f`, чтобы определить `pid` нашей программы-нагрузчика.

В итоге получим:

```
$ perf stat ./ema_search ./text/out.txt
QalIVKumQfiPgyJrzwoQMmPlmtfUcM 1000
```

```
top - 09:57:28 up 1 day,  2:14,  0 users,  load average: 1.20, 0.71, 0.54
Tasks:   1 total,   1 running,   0 sleeping,   0 stopped,   0 zombie
%Cpu(s): 11.6 us,  1.0 sy,  0.0 ni, 87.3 id,  0.0 wa,  0.0 hi,  0.1 si,  0.0 st
MiB Mem :   3923.1 total,   2086.1 free,    560.2 used,   1276.9 buff/cache
MiB Swap:   1024.0 total,   1024.0 free,      0.0 used.   3165.7 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  195 root      20   0    2188   1280   1280 R  99.7   0.0   1:29.41 ema_search

>>> Total execution time: 95.629390 seconds <<<


 Performance counter stats for './bin/ema_search text/out.txt QalIVKumQfiPgyJrzwoQ 1000':

         95616.78 msec task-clock                #    0.990 CPUs utilized
             4849      context-switches          #   50.713 /sec
              209      cpu-migrations            #    2.186 /sec
               52      page-faults               #    0.544 /sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

      96.605819461 seconds time elapsed

      88.346751000 seconds user
       7.288569000 seconds sys
```

Как можно заметить, один процесс нагружает одно ядро почти на 100%.

Для увеличения количества нагрузчиков воспользуемся скриптом `parallel.sh`

Увеличим число процессов вдвое, получим:

```
$ $ bash ./parallel.sh 2 ./bin/ema_search ./text/out.txt
QalIVKumQfiPgyJrzwoQMmPlmtfUcM 1000
```

```
top - 14:53:10 up 17:43,  0 users,  load average: 2.52, 1.74, 1.20
Tasks:   9 total,   5 running,   4 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.3 us,  0.0 sy,  0.0 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu1  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :  0.7 us,  0.0 sy,  0.0 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :  0.3 us,  0.0 sy,  0.0 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.3 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
MiB Mem :   3923.1 total,   2111.6 free,    565.9 used,   1245.6 buff/cache
MiB Swap:   1024.0 total,   1024.0 free,      0.0 used.   3161.7 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  647 root      20   0    6096   5276   1280 R 100.0   0.1   0:31.75 sort
  645 root      20   0    6096   5276   1280 R  99.7   0.1   0:31.74 sort
  646 root      20   0    6096   5276   1280 R  99.7   0.1   0:31.74 sort
  648 root      20   0    6096   5336   1280 R  99.7   0.1   0:31.74 sort
    1 root      20   0    4116   2944   2688 S   0.0   0.1   0:00.02 bash
    9 root      20   0    4636   3840   2816 S   0.0   0.1   0:01.37 bash
  501 root      20   0    4120   3200   2816 S   0.0   0.1   0:00.11 bash
  644 root      20   0    3852   2816   2560 S   0.0   0.1   0:00.00 bash
  649 root      20   0    6864   2944   2432 R   0.0   0.1   0:00.02 top


All 2 instances have finished.

 Performance counter stats for 'bash ./parallel.sh 2 ./bin/ema_search ./text/out.txt QalIVKumQfiPgyJrzwoQMmPlmtfUcM 1000':

        185069.73 msec task-clock                #    1.879 CPUs utilized
           186127      context-switches          #    1.006 K/sec
              900      cpu-migrations            #    4.863 /sec
              538      page-faults               #    2.907 /sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

     98.515731586 seconds time elapsed

    173.613493000 seconds user
     11.976179000 seconds sys
```

Увеличим число процессов втрое, получим:

```
$ perf stat bash ./parallel.sh 3 ./bin/ema_search
QalIVKumQfiPgyJrzwoQMmPlmtfUcM 1000
```

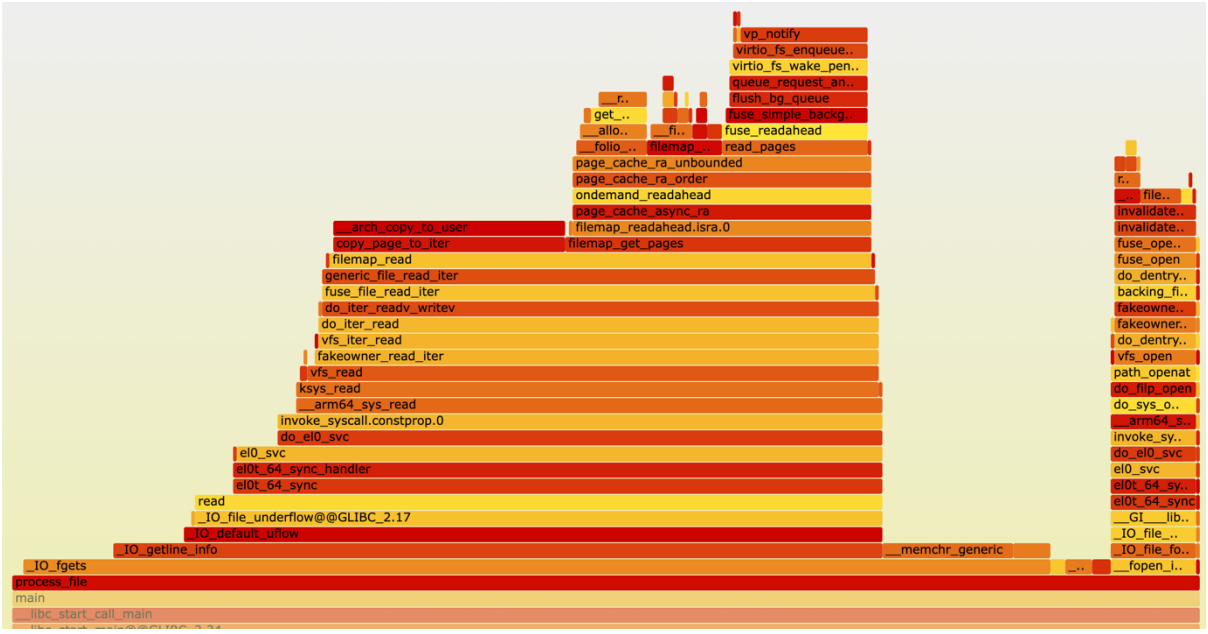
```
top - 11:14:37 up 1 day,  3:14,  0 users,  load average: 2.05, 1.61, 2.62
Tasks:   9 total,   4 running,   5 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.3 us,  1.7 sy,  0.0 ni, 95.9 id,  0.0 wa,  0.0 hi,  2.0 si,  0.0 st
%Cpu1  : 28.2 us,  1.7 sy,  0.0 ni, 67.8 id,  1.7 wa,  0.0 hi,  0.7 si,  0.0 st
%Cpu2  : 13.0 us,  0.7 sy,  0.0 ni, 85.4 id,  0.7 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu3  : 48.0 us,  3.3 sy,  0.0 ni, 45.3 id,  3.3 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  : 35.1 us,  2.3 sy,  0.0 ni, 59.9 id,  2.3 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu5  : 39.7 us,  3.3 sy,  0.0 ni, 53.7 id,  3.0 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu6  : 50.5 us,  3.7 sy,  0.0 ni, 42.5 id,  3.3 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  : 50.0 us,  3.3 sy,  0.0 ni, 44.0 id,  2.7 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3923.1 total,   2086.4 free,    560.1 used,   1276.6 buff/cache
MiB Swap:   1024.0 total,   1024.0 free,      0.0 used.   3165.5 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  351 root      20   0    2188   1280   1280 R  93.7   0.0   0:39.95 ema_search
  352 root      20   0    2188   1280   1280 R  93.7   0.0   0:39.96 ema_search
  350 root      20   0    2188   1280   1280 R  93.4   0.0   0:39.95 ema_search
    1 root      20   0    4116   2944   2688 S   0.0   0.1   0:00.01 bash
    9 root      20   0    4120   3200   2816 S   0.0   0.1   0:01.25 bash
   42 root      20   0    4120   3328   2816 S   0.0   0.1   0:00.18 bash
  341 root      20   0    6864   2944   2432 R   0.0   0.1   0:00.23 top
  348 root      20   0   28352   9984   8320 S   0.0   0.2   0:00.01 perf
  349 root      20   0    3852   2816   2560 S   0.0   0.1   0:00.00 bash

All 3 instances have finished.

 Performance counter stats for 'bash ./parallel.sh 3 ./bin/ema_search ./text/out.txt QalIVKumQfiPgyJrzwoQMmPlmtfUcM 1000':

        278944.63 msec task-clock                #    2.832 CPUs utilized
           311631      context-switches          #    1.117 K/sec
             2022      cpu-migrations            #    7.249 /sec
              672      page-faults               #    2.409 /sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

     98.489336127 seconds time elapsed

    262.977625000 seconds user
     16.816881000 seconds sys
```

Увеличим число процессов вчетверо, получим:

```
$ perf stat bash ./parallel.sh 4 ./bin/ema_search
QalIVKumQfiPgyJrzwoQMmPlmtfUcM 1000
```

```
top - 11:17:32 up 1 day,  3:16,  0 users,  load average: 2.75, 1.95, 2.58
Tasks:  10 total,   5 running,   5 sleeping,   0 stopped,   0 zombie
%Cpu0  :  3.4 us,  2.4 sy,  0.0 ni, 92.2 id,  0.3 wa,  0.0 hi,  1.7 si,  0.0 st
%Cpu1  : 36.7 us,  4.0 sy,  0.0 ni, 57.3 id,  2.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  : 53.7 us,  5.3 sy,  0.0 ni, 38.0 id,  2.7 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu3  : 39.3 us,  3.7 sy,  0.0 ni, 55.4 id,  1.7 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  : 50.3 us,  4.7 sy,  0.0 ni, 42.7 id,  2.3 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  : 60.8 us,  5.6 sy,  0.0 ni, 30.9 id,  2.7 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  : 47.0 us,  4.7 sy,  0.0 ni, 46.3 id,  2.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  : 59.2 us,  5.7 sy,  0.0 ni, 32.8 id,  2.3 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3923.1 total,   2073.3 free,    560.8 used,   1289.0 buff/cache
MiB Swap:   1024.0 total,   1024.0 free,      0.0 used.   3164.5 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  356 root      20   0    2188   1280   1280 R  95.7   0.0   0:41.35 ema_search
  357 root      20   0    2188   1280   1280 R  95.3   0.0   0:41.35 ema_search
  358 root      20   0    2188   1280   1280 R  95.3   0.0   0:41.36 ema_search
  359 root      20   0    2188   1280   1280 R  95.3   0.0   0:41.35 ema_search
  341 root      20   0    6864   2944   2432 R   0.3   0.1   0:00.41 top
    1 root      20   0    4116   2944   2688 S   0.0   0.1   0:00.01 bash
All 4 instances have finished.

 Performance counter stats for 'bash ./parallel.sh 4 ./bin/ema_search ./text/out.txt QalIVKumQfiPgyJrzwoQMmPlmtfUcM 1000':

        391903.22 msec task-clock                #    3.787 CPUs utilized
           400396      context-switches          #    1.022 K/sec
             3315      cpu-migrations            #    8.459 /sec
              808      page-faults               #    2.062 /sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

     103.476134714 seconds time elapsed

     358.549516000 seconds user
      34.341158000 seconds sys
```

Построим Flame Graph для программы-нагрузчика:



Вывод проделанных измерений

При увеличении числа процессов одновременно нагружается не одно ядро, а сразу несколько. Также было замечено, что время выполнения увеличивалось на несколько секунд по сравнению с предыдущим шагом.

Оптимизации компилятора

Sort

Для сравнения запустим `sort` 1000 и возьмем массив длиной 100000

```
# gcc ./src/sort.c -o ./bin/sort -O0
# perf stat ./bin/sort 100000 1000

>>> Total execution time: 18.272818 seconds <<<


 Performance counter stats for './bin/sort 100000 1000':

         18276.53 msec task-clock                #    1.000 CPUs utilized
               32      context-switches          #    1.751 /sec
                0      cpu-migrations            #    0.000 /sec
              149      page-faults               #    8.153 /sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

      18.279176758 seconds time elapsed

      18.266860000 seconds user
       0.011001000 seconds sys
```

Теперь скомпилируем `sort` с уровнем оптимизации `-O3`

```
# gcc ./src/sort.c -o ./bin/sort -O3
# perf stat ./bin/sort 100000 1000
```

```
>>> Total execution time: 6.995622 seconds <<<


 Performance counter stats for './bin/sort 100000 1000':

          6998.28 msec task-clock                #    1.000 CPUs utilized
                3      context-switches          #    0.429 /sec
                0      cpu-migrations            #    0.000 /sec
              148      page-faults               #   21.148 /sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

       7.000473504 seconds time elapsed

       6.995955000 seconds user
       0.003006000 seconds sys
```

При добавлении агрессивной оптимизации для компилятора время выполнения программы уменьшилось в 2.6 раза

Ema Search String

Для сравнения запустим `search` 1000 раз и будем искать подстроку `QalIVKumQfiPgyJrzwoQMmPlmtfUcM`

```
# gcc ./src/ema_search.c -o ./bin/ema_search -O0
# perf stat ./bin/ema_search ./text/out.txt QalIVKumQfiPgyJrzwoQMmPlmtfUcM 1000
>>> Total execution time: 93.215281 seconds <<<


 Performance counter stats for './bin/ema_search ./text/out.txt QalIVKumQfiPgyJrzwoQMmPlmtfUcM 1000':

         93215.44 msec task-clock                #    0.986 CPUs utilized
            11888      context-switches          #  127.533 /sec
              231      cpu-migrations            #    2.478 /sec
               50      page-faults               #    0.536 /sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

      94.494693544 seconds time elapsed

      86.082916000 seconds user
       7.134628000 seconds sys
```

Теперь скомпилируем `ema_search` с уровнем оптимизации `-O3`

```
# gcc ./src/ema_search.c -o ./bin/ema_search -O3
# perf stat ./bin/ema_search ./text/out.txt QalIVKumQfiPgyJrzwoQMmPlmtfUcM 1000
```

```
>>> Total execution time: 38.119794 seconds <<<


 Performance counter stats for './bin/ema_search ./text/out.txt QalIVKumQfiPgyJrzwoQMmPlmtfUcM 1000':

         38103.32 msec task-clock                #    0.975 CPUs utilized
             6184      context-switches          #  162.296 /sec
              251      cpu-migrations            #    6.587 /sec
               50      page-faults               #    1.312 /sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

     39.066416018 seconds time elapsed

     31.399403000 seconds user
      6.721936000 seconds sys
```

При добавлении агрессивной оптимизации для компилятора время выполнения программы уменьшилось в 2.4 раза

Вывод

В ходе выполнения лабораторной работы я разработал свою оболочку shell и узнал, как можно создавать клонировать процессы в Linux. Также разработал комплекс программ-нагрузчиков познакомился с основными утилитами для профилирования.