PW - laboratorium 5
Michał Mirek 125637
Wtorek 11:00

Wstęp

Laboratorium ma na celu:

- ✦ zastosowanie współbieżności dla algorytmów

Repozytorium GitHub z kodem: https://github.com/mmirek98/cc-classes/tree/lab-05 (dla każdego zadania wymagającego modyfikacji plików źródłowych istnieje odpowiadający commit z wiadomością, np. “ex-3” to zmiany dla zadania 3).

Zadanie 1

Posortować ciąg liczb wykorzystując algorytm scalania dwóch posortowanych ciągów (merge sort). Zadany ciąg podzielono na dwie części. Każda z nich jest sortowana oddzielnie, następnie posortowane części są scalane. Sortowanie obu połówek oraz procedura scalająca wykonywane mają być współbieżnie.

Ze zdobytą wiedzą z poprzednich laboratorium, możemy zrekonstruować algorytm sortowania przez scalanie z użyciem wielowątkowości. Każdy wątek będzie odpowiedzialny za posortowanie tablicy w zakresie określonym poprzez parametry funkcji wątku *mergeSortConc*. Jeśli zakres jest większy od zera, to wątek dzieli swój zakres na dwie części i tworzy kolejne wątki, rekurencyjnie wywołując funkcję *mergeSortConc*. Następnie wywołuje funkcję scalającą *mergeConc*.

Poniżej przedstawiono fragmenty programy (całość zawiera także algorytm sortowania synchronicznego w ramach porównania)

sort.c

```
struct Borders {
 int start;
 int end;
};

int array[LEN];

// does actual sorting
void *mergeConc(void* bord) {
 struct Borders borders = *((struct Borders *) bord);
 int center = (borders.start + borders.end) / 2;
 int leftIt = borders.start;
 int rightIt = center + 1;
 int currentPos = 0;
 int tmpTab[LEN];

 while (leftIt <= center && rightIt <= borders.end) {
   if (array[leftIt] < array[rightIt]) {
     tmpTab[currentPos] = array[leftIt];
     leftIt++;
     currentPos++;
   } else {
     tmpTab[currentPos] = array[rightIt];
```

```
    rightIt++;
    currentPos++;
   }

 }

 if (leftIt <= center) {
   while (leftIt <= center) {
     tmpTab[currentPos] = array[leftIt];
     leftIt++; currentPos++;
   }
 } else if (rightIt <= borders.end) {
   while (rightIt <= borders.end) {
     tmpTab[currentPos] = array[rightIt];
     rightIt++; currentPos++;
   }
 }

 currentPos = 0;
 int i;
 for (i = borders.start; i <= borders.end; i++) {
   array[i] = tmpTab[currentPos];
   currentPos++;
 }
}

// doesn't care about array, only divide
void *mergeSortConc(void* bord) {
 struct Borders borders = *((struct Borders *) bord);
 struct Borders left, right;
 pthread_t leftTid;
 pthread_t rightTid;

 left.start = borders.start;
 left.end = (borders.start + borders.end) / 2;
 right.start = left.end + 1;
 right.end = borders.end;

 if (borders.start < borders.end) {
   pthread_create(&leftTid, NULL, mergeSortConc, &left);
   pthread_create(&rightTid, NULL, mergeSortConc, &right);
   pthread_join(leftTid, NULL);
   pthread_join(rightTid, NULL);
   mergeConc(&borders);
 }
}

int main() {
 struct Borders borders;
 borders.start = 0;
 borders.end = (sizeof array / sizeof *array) - 1;
 prepareArray();

 clock_t start = clock();
 mergeSortConc(&borders);
 clock_t end = clock();

 printf("\n\n");
 printf("Czas wykonywania dla wspolbieznego: %lu ms\n", end-start);
}
```

Wynik:

```
694     94      11      505     619     255     42      717     734     68
607     103     869     885     492     79      27      438     342     595
954     902     128     82      780     732     804     634     991     835
301     685     929     664     543     900     920     585     617     654
5       224     109     226     461     601     306     489     39      0
436     993     254     564     428     35      648     584     669     640
419     322     325     701     986     868     601     906     805     571
912     811     147     21      37      609     974     695     450     14
696     886     359     950     450     139     985     99      723     654
739     143     976     416     196     315     637     797     221     442
720     486     605     868     507     643     829     482     690     279
848     386     165     559     689     615     699     26      66      422
681     157     917     9       574     113     324     563     263     898
5       983     736     963     203     243     958     32      77      648
311     277     387     476     837     76      444     536     102     862
310     135     20      228     497     946     341     173     509     604
423     866     940     159     829     143     755     139     176     832
788     839     110     175     316     299     603     112     835     57
974     145     545     994     725     42      292     67      567     153
23      343     20      315     502     201     459     609     341     987



0       5       5       9       11      14      20      20      21      23
26      27      32      35      37      39      42      42      57      66
67      68      76      77      79      82      94      99      102     103
109     110     112     113     128     135     139     139     143     143
145     147     153     157     159     165     173     175     176     196
201     203     221     224     226     228     243     254     255     263
277     279     292     299     301     306     310     311     315     315
316     322     324     325     341     341     342     343     359     386
387     416     419     422     423     428     436     438     442     444
450     450     459     461     476     482     486     489     492     497
502     505     507     509     536     543     545     559     563     564
567     571     574     584     585     595     601     601     603     604
605     607     609     609     615     617     619     634     637     640
643     648     648     654     654     664     669     681     685     689
690     694     695     696     699     701     717     720     723     725
732     734     736     739     755     780     788     797     804     805
811     829     829     832     835     835     837     839     848     862
866     868     868     869     885     886     898     900     902     906
912     917     920     929     940     946     950     954     958     963
974     974     976     983     985     986     987     991     993     994


Czas wykonywania dla asynchronicznego: 62 ms
```

Zadanie 3

Napisać program znajdujący liczby pierwsze używając tzw. sita Erastotenesa.

Dla ustalonej liczby N znajdujemy liczby pierwsze. Aby tego dokonać będziemy iterować po liczbach, które są mniejsze bądź rowne pierwiastkowi kwadratowemu z liczby N. Dla każdej liczby utworzymy wątek, który będzie oznaczał wielokrotności danej liczby jako liczby nie pierwsze (0 w tablicy *prime* oznacza liczbę złożoną, 1 - to liczba pierwsza). Musimy uchronić się przed *race condition,* zatem użyjemy muteksów w celu dostępu do sekcji krytycznej (tablicy *primes*).

```c
#define TOTAL 1000

int prime[TOTAL];
pthread_mutex_t shield;

void* markNonPrimes(void* i) {
 int j = *((int*)i);
  pthread_mutex_lock(&shield);
 if (prime[j] == 1) {
   int k;
   for (k = j*j; k < TOTAL; k += j) {
   prime[k] = 0;
   pthread_mutex_unlock(&shield);
   }
 } else {
   pthread_mutex_unlock(&shield);
 }
 pthread_exit(NULL);
}

void sieveConc() {
 int i =0;
 for (i = 0; i < TOTAL; i++) {
   prime[i] = 1;
 }
 int j;
 int size = sqrt(TOTAL);
 pthread_t pool[size];
 int it = 0;

 for (j = 2; j*j < TOTAL; j++) {
   pthread_create(&pool[it], NULL, markNonPrimes, &j);
   it++;
 }

 it = 0;
 for (j = 0; j < size; j++) {
   pthread_join(pool[it], NULL);
   it++;
 }

 j = 0;
 for (j = 0; j <= TOTAL; j++) {
   if (prime[j] == 1) {
     printf("%d\t", j);
   }
 }
}

void prepareArray() {
 int i;
 for (i = 0; i < TOTAL; i++) {
   prime[i] = 1;
 }
}
int main() {
 prepareArray();
 sieveConc();
  return 0;
}
```

Wynik programu: