Część II
Podział pracy

1 Tablica sortująca

Kolejka priorytetowa to struktura danych udostępniająca operacje wstawienia wartości i pobrania wartości minimalnej. Z kolejki liczb całkowitych, za pośrednictwem funkcji

```
void wstaw(int x);
int usuńMin(int i);
```

korzystają procesy:

```
process P[id : 1..K] {
    int x;
    ...
    wstaw(x);
    ...
    x = usuńMin(id);
    ...
}
```

Zakładamy, że próba pobrania elementu z kolejki pustej lub wstawienia do kolejki pełnej powinna powodować zawieszenie procesu w oczekiwaniu na możliwość wykonania tej operacji.

Zrealizuj kolejkę priorytetową liczb całkowitych przechowującą maksymalnie N elementów. Implementacja powinna się składać z funkcji wstaw i usuńMin oraz tablicy połączonych w "rurociąg" procesów pomocniczych, przechowujących po jednej liczbie.

Rozwiązanie

```
process P[id : 1..K] {
    int x;
    ...
    wstaw(x);
    ...
    x = usuńMin(id);
    ...
}

void wstaw(int x) {
    send Q[1].x;
}

int usuńMin(int i) {
    int x;
    send Q[1].daj(i);
    receive x;
    return x;
}
```

```
process Q[id : 1..N] {
    int ile = 0;
    int moja, nowa, k;
    while (true) select {
        if (ile < N - id + 1) on nowa
            if (++ile == 1)
                moja = nowa;
            else {
                if (moja > nowa) {
                    int s = moja;
                    moja = nowa;
                    nowa = s;
                }
                send Q[id + 1].nowa;
            }
        if (ile > 0) on daj(k) {
            if (id == 1)
                send P[k].moja;
            else
                send Q[k].moja;
            if (--ile > 0) {
                send Q[id + 1].daj(id);
                receive moja;
            }
        }
    }
}
```

2 Firma

W pewnej firmie pracuje N pracowników. Poziom umiejętności każdego z nich jest wyrażony dodatnią liczbą całkowitą, którą pracownik poznaje, wywołując funkcję

```
int mójPoziom();
```

Firma obsługuje zlecenia K klientów, reprezentowane przez wartości typu zlecenie. Jedno zlecenie jest wykonywane przez grupę pracowników, których suma poziomów umiejętności jest równa co najmniej $P > 0$, a poziom każdego pracownika jest mniejszy od P. Pracownicy przydzieleni do zlecenia wykonują je sekwencyjnie, zgodnie z kolejnością, w jakiej zgłaszali się do pracy, wywołując funkcję

```
void przetwórz(zlecenie &z);
```

Pierwszy pracownik dostaje zlecenie od klienta i, po przetworzeniu, przekazuje je dalej. Ostatni pracownik, po odebraniu i przetworzeniu zlecenia, oddaje je klientowi. Klient wykonuje algorytm podany w rozwiązaniu. Firma chce obsłużyć jak najwięcej zleceń klientów, w związku z czym gorliwy pracownik może uczestniczyć parokrotnie w przetwarzaniu jednego zlecenia.

Zapisz treść procesów Pracowników oraz Firmy zarządzającej ich pracą.

Rozwiązanie

```
type zlecenie;
void inicjuj(zlecenie &z);
void konsumuj(zlecenie z);
int mójPoziom();
void przetwórz(zlecenie &z);

process Klient[id : 1..K] {
    zlecenie z;
    int pierwszy;
    while (true) {
        inicjuj(z);
        send Firma.zgłoszenieKlienta(id);
        receive pierwszy;
        send Pracownik[pierwszy].z;
        receive z;
        konsumuj(z);
    }
}

process Pracownik[id : 1..N] {
    zlecenie z;
    int komu;
    bool jestOstatni;
    int poziom = mójPoziom();

    while (true) {
        send Firma.zgłoszeniePracownika(id, poziom);
        receive (komu, jestOstatni);
        receive z;
        przetwórz(z);
        if (jestOstatni)
            send Klient[komu].z;
        else
            send Pracownik[komu].z;
    }
}

process Firma {              // wersja bardziej współbieżna
  int suma, nk, np, poziom, pierwszy;

  while (true) {
    suma = 0;
    poprzedni = 0;           // brak pracownika
    nk = 0;                  // brak klienta
    while ((suma < P) && (nk == 0)) {
      select {
        if (nk == 0) on zgłoszenieKlienta(nk) {
           if (poprzedni != 0) send Klient[nk].pierwszy;
        }
        if (suma < P) on zgłoszeniePracownika(np, poziom) {
           suma += poziom;
           if (poprzedni == 0) {     // pierwszy pracownik
```

```
                pierwszy = np;
                if (nk > 0) send Klient[nk].pierwszy;
            }
            else
                send Pracownik[poprzedni].(np, false);
            poprzedni = np;
        }
      }
    }
    send Pracownik[poprzedni].(nk, true);    // ostatni
  }
}

process Firma {                          // wersja zeszłoroczna
    int suma, nk, np, poziom;            //  (żywcem z Ady)
    while (true) {
        suma = 0;
        poprzedni = 0;
        receive zgłoszenieKlienta(nk);
        while (suma < P) {
            receive zgłoszeniePracownika(np, poziom);
            suma += poziom;
            if (poprzedni == 0) {
                send Klient[nk].np;
            } else
                send Pracownik[poprzedni].(np, false);
            poprzedni = np;
        }
        send Pracownik[poprzedni].(nk, true);
    }
}
```

3 Gra w Życie

Gra w Życie (ang. Life) to automat komórkowy w dwuwymiarowym świecie. Komórki zorganizowane są w torus, który ma $W > 1$ wierszy o numerach $0 \,..\, W-1$ i $K > 2$ kolumn o numerach $0 \,..\, K-1$. Zakładamy, że W i K są parzyste. Przyjmujemy, że komórka o współrzędnych (a, b) sąsiaduje z komórką (c, d), jeśli a i c różnią się, modulo W, nie więcej niż o 1 oraz b i d różnią się, modulo K, nie więcej niż o 1. Z każdą komórką sąsiaduje więc osiem innych.

Komórki moga być w jednym z dwóch stanów — żywe lub martwe. Łączny stan wszystkich komórek świata nazywamy generacją. W kolejnej generacji komórka jest żywa wtedy i tylko wtedy, gdy:

- w poprzedniej generacji była żywa i miała dwóch lub trzech żywych sąsiadów albo,
- w poprzedniej generacji była martwa i miała trzech żywych sąsiadów.

Proces Życie czyta początkową generację. Następnie, korzystając z tablicy procesów Komórka, wyznacza generację numer N i wypisuje ją. Podaj treść procesów Komórka przy założeniu, że proces Życie wykonuje algorytm podany w rozwiązaniu.

Rozwiązanie

```
void czytaj(bool[][] &świat);
void pisz(bool[][] świat);

process Życie {
    bool[][] świat;
    int i, j, ai, aj;
    czytaj(świat);
    for (i = 0; i < W; ++i)
        for (j = 0; j < K; ++j)
            send Komórka[i][j].świat[i][j];
    for (i = 0; i < W * K; ++i) {
        bool x;
        receive (ai, aj, x);
        świat[ai][aj] = x;
    }
    pisz(świat);
}

int wymien(int di, int dj, int e, bool w) {
    int z = 0;
    for (int y = 0; y < 2; ++y)
        for (int s = 0; s < 4; ++s)
            if (y == e) {          // ja najpierw wysyłam
                send Komórka[(i + di + W) % W]
                            [(j + dj + K) % K].w;
                int d = di;
                di = dj;
                dj = -d;
            } else {             // a ja najpierw odbieram
                bool x;
                receive x;
                if (x)
                    ++z;
            }
    return z;
}

process Komórka[i : 0..W-1][j : 0..K-1] {
    bool w;
    receive w;
    for (int g = 0; g < N; ++g) {
        int z = wymien(1, 0, (i + j) % 2, w);
        z += wymien(1, 1, i % 2, w);
        w = (z == 3) || (z == 2 && w);
    }
    send Życie.(i, j, w);
}
```

4 Mnożenie macierzy

System składający się z procesu Serwer oraz N > 0 procesów Pracownik liczy kwadraty wczytywanych macierzy rozmiaru K x K, gdzie K > 1 i K * K <= N. Proces Serwer działa w nieskończonej pętli. Wczytuje w niej, funkcją

```
void czytaj(float [][] &m);
```

macierz M o rozmiarze K x K. Zgłaszającym się do niego procesom Pracownik przekazuje po jednym elemencie macierzy M. Następnie odbiera od nich elementy kwadratu macierzy M i, po zebraniu wszystkich, wypisuje wynik funkcją

```
void pisz(float [][] m);
```

Pracownik, również w nieskończonej pętli, odpoczywa wykonując

```
void własneSprawy();
```

a następnie zgłasza Serwerowi gotowość wykonania obliczeń. Otrzymuje od niego jeden element macierzy M, z wiersza i, kolumny j a od pozostałych pracowników zdobywa wszystkie elementy z tego wiersza i kolumny. Następnie liczy wartość z wiersza i kolumny j w macierzy wynikowej i przekazuje ją do Serwera.

Zapisz treści procesów Serwer i Pracownik. Postaraj się zminimalizować rozmiar pamięci każdego z procesów, liczbę przesyłanych komunikatów i ich rozmiar.

Rozwiązanie

```
#define N 1000
#define K 20

void własneSprawy();
void czytaj(float [][] &m);
void pisz(float [][] m);

void rozdaj() {
    int i, j, n;
    int ostatniWiersz[K];
    int ostatniaKolumna[K];
    int poprzedni[K];
    for (i = 0; i < K; ++i) {
        receive zgłaszamSię(n);
        ostatniWiersz[i] = poprzedni[i] = n;
    }
    for (i = 0; i < K - 1; ++i)
        receive zgłaszamSię(ostatniaKolumna[i]);
    ostatniaKolumna[K - 1] = ostatniWiersz[K - 1];
    for (i = K - 2; i >= 0; --i) {
        poprzedni[K - 1] = ostatniaKolumna[i];
        for (j = K - 2; j >= 0; --j) {
            receive zgłaszamSię(n);
            send Pracownik[n].(i, j, poprzedni[j],
                    poprzedni[j + 1], m[i][j]);
            poprzedni[j] = n;
        }
```

```
        send Pracownik[ostatniaKolumna[i]].(i, K - 1,
                n, ostatniaKolumna[i + 1], m[i][K - 1]);
    }
    for (j = 0; j < K - 1; ++j)
        send Pracownik[ostatniWiersz[j]].(K - 1, j,
                poprzedni[j], ostatniWiersz[j + 1],
                m[K - 1][j]);
    send Pracownik[ostatniWiersz[K - 1]].(K - 1, K - 1,
            poprzedni[K - 1], poprzedni[0],
            m[K - 1][K - 1]);
}

process Serwer {
    while (true) {
        float[][] m;
        czytaj(m);
        rozdaj(m);
        for (int i = 0; i < K * K; ++i) {
            int ai, aj;
            float w;
            receive (ai, aj, w);
            m[ai][aj] = w;
        }
        pisz(m);
    }
}

process Pracownik[n : 1..N] {
    int i, j, ni, nj;
    float moja;
    float[] wiersz, kolumna;
    bool mamWiersz, mamKolumnę;
    while (true) {
        własneSprawy();
        send Serwer.zgłaszamSię(n);
        receive (i, j, ni, nj, moja);
        if (i == 0) {
            float nowaKolumna[K];
            nowaKolumna[0] = moja;
            send Pracownik[nj].
                    budowanaKolumna(nowaKolumna);
        }
        if (j == 0) {
            float nowyWiersz[K];
            nowyWiersz[0] = moja;
            send Pracownik[ni].
                    budowanyWiersz(nowyWiersz);
        }
        mamWiersz = false;
        mamKolumnę = false;
        while (!(mamWiersz && mamKolumnę)) select {
            on budowanyWiersz(wiersz) {
```

```
                wiersz[j] = moja;
                if (j < K - 1) {
                    send Pracownik[nj].
                            budowanyWiersz(wiersz);
                } else
                    send Pracownik[nj].
                            gotowyWiersz(wiersz);
            }
            on budowanaKolumna(kolumna) {
                kolumna[i] = moja;
                if (i < K - 1) {
                    send Pracownik[ni].
                            budowanaKolumna(kolumna);
                } else
                    send Pracownik[ni].
                            gotowaKolumna(kolumna);
            }
            on gotowyWiersz(wiersz) {
                mamWiersz = true;
                if (j < K - 1)
                    send Pracownik[nj].
                            gotowyWiersz(wiersz);
            }
            on gotowaKolumna(kolumna) {
                mamKolumnę = true;
                if (i < K - 1)
                    send Pracownik[ni].
                            gotowaKolumna(kolumna);
            }
        }
        wynik = 0.0;
        for (int p = 0; p < K; ++p)
            wynik += wiersz[p] * kolumna[p];
        send Serwer.(i, j, wynik);
    }
}
```