Ταυτόχρονος Προγραμματισμός

ΣΕΙΡΆ ΕΡΓΑΣΙΏΝ 3

Άσκηση 1 – Σηματοφόροι στο πνεύμα ενός ελεγκτή

Στην βιβλιοθήκη υπάρχουν οι συναρτήσεις:

```
int mysem_init(mysem_t* s, int n);
int mysem_up(mysem_t s*);
int mysem_down(mysem_t s*);
int mysem_destroy(mysem_t s*);
```

Το struct mysem_t περιεχει τα εξης πεδια:

1) Int val <-- Τιμη του σηματοφορου
2) Int id <-- Αναγνωριστικο του σηματοφορου
3) Int wait <-- Αριθμος διεργασιων που περιμενουν για σημα up
4) Bool init <-- Τιμες 0 ή 1 , 1 για το αν εχει αρχικοποιηθει 0 αλλιως
5) pthread_mutex_t mutex <-- Μηχανισμος για τον ελενκτη
6) Pthread_cond_t cond <-- Μεταβλητη συνθηκης χρησιμη για τον συγχρονισμο

Τέλος, έχουμε ένα static int counter που χρησιμοποιείται για την σωστή αρχικοποίηση και καταστροφή ενός "σηματοφόρου".

int mysem_init(mysem_t* s, int n)

```
/* Έλεγχος αν ο σηματοφόρος έχει ήδη αρχικοποιηθεί:
 if (s->init == true) return (-1);*/
/* Έλεγχος αν δίνουμε λανθασμένη αρχικοποιημένη τιμή (έλεγχος του n):
 if (n ≠ 0,1) return (0); */
/* s->mutex & s->cond init */
/*begin monitor region*/
s->val = n;
s->id = counter;
s->init = true;
s->wait = 0;
counter++;
/*end monitor region*/
return (1);
```

int mysem_up(mysem_t s*)

```
/* Έλεγχος αν ο σηματοφόρος έχει αρχικοποιηθεί:
 if (s->init == false) return (-1);*/
/*begin monitor region*/
if (s->val == 1) {
    if (s->wait > 0)    //περιμένουν στο queue του σηματοφόρο
        for(i = 0; i < s->wait; i++) { signal(s->cond); } // Εαν υπαρχουν νηματα σε down τοτε ξυπνησε τα
    /*end monitor region*/
    return (0);
}
s->val++;    //Αλλιώς (αν είναι 0), να γίνει 1
s->wait--; // Μειωσε τον αριθμο των νηματων που περιμενουν
signal(s->cond); // Δωσε σημα για ξυπνημα
/*end monitor region*/
return (1);
```

int mysem_down(mysem_t s*)

```
/* Έλεγχος αν ο σηματοφόρος έχει αρχικοποιηθεί:
 if (s->init == false) return (-1);*/
/*begin monitor region*/
while(s->val == 0) {    //loop στο οποίο γίνεται η αναμονή του νήματος
    s->wait++; // Αυξησε την τιμη των νηματων που περιμενουν
    wait(s->cond);
    s->wait--; // Μειωσε την τιμη των νηματων που περιμενουν
}
If (s->val == 1) {    //Η τιμή σηματοφόρου μειώνεται μόνο αν είναι ίση με 1
    s->val--;
}
/*end monitor region*/
return (1);
```

int mysem_destroy(mysem_t s*)

```
/* Έλεγχος αν ο σηματοφόρος έχει αρχικοποιηθεί:
 if (s->init == false or (s->id > counter or s->id < 1)) return (-1);*/
/*begin monitor region*/
If (s->wait != 0) {
    for(i = 0; i < s->wait; i++) { signal(s->cond); } // Σηματοδοτισε το ξυπνημα οσων νηματων δεν εχουν γινει up και περιμενουν σε down για ομαλη έξοδο
}
/*end monitor region*/
/* destroy s->cond & s->mutex */
s->init = false;
return (1);
```

Άσκηση 2 -Ψευδοκώδικας

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 0]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 0]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 0]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 0]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 0]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 0]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 0]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Έστω μετά το signal συνεχίζει ο worker!

Άσκηση 2 -Κανονική λειτουργία [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 0]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 0]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

*(Αν είχαμε signal-continue πριν θα ξεμπλόκαρε η main!)

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 0]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 0]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 0]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 0]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 0]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 0]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Φτάσαμε στην (περίπου) αρχική κατάσταση!

Άσκηση 2 -Κανονική λειτουργία [count = 0]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

*Περίπτωση που ο worker δεν παίρνει το mutex

Άσκηση 2 -Κανονική λειτουργία [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

*Περίπτωση που ο worker δεν παιρνει το mutex

Άσκηση 2 -Κανονική λειτουργία [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

*Περίπτωση που ο worker δεν παιρνει το mutex

Άσκηση 2 -Κανονική λειτουργία [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

*Περίπτωση που ο worker δεν παιρνει το mutex

Άσκηση 2 -Κανονική λειτουργία [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

*Περίπτωση που ο worker δεν παιρνει το mutex

Άσκηση 2 -Κανονική λειτουργία [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

*Περίπτωση που ο worker δεν παίρνει το mutex

Άσκηση 2 -Κανονική λειτουργία [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

*Περίπτωση που ο worker δεν παιρνει το mutex

Άσκηση 2 -Κανονική λειτουργία [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

*Περίπτωση που ο worker δεν παιρνει το mutex

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 -Κανονική λειτουργία [count = 0]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

*Περίπτωση που ο worker δεν παιρνει το mutex

Άσκηση 2 -Κανονική λειτουργία [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

*Περίπτωση που ο worker δεν παιρνει το mutex

Άσκηση 2 -Κανονική λειτουργία [count = 0]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

*Περίπτωση που ο worker δεν παιρνει το mutex

Άσκηση 2 -Κανονική λειτουργία [count = 0]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

*Περίπτωση που ο worker δεν παιρνει το mutex

Άσκηση 2 -Κανονική λειτουργία [count = 0]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

Όλα καλά!

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 –Τερματισμός [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 –Τερματισμός [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 –Τερματισμός [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 –Τερματισμός [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 –Τερματισμός [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 –Τερματισμός [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 –Τερματισμός [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 –Τερματισμός [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 –Τερματισμός [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 –Τερματισμός [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 0 && terminate != true) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        if(terminate == true) break;
        prime = *(int *)arg;
        count -= 1;

        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);

      *calculate if prime
       }
terminated += 1;
pthread_mutex_unlock(&mutex);
return(NULL); }
```

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 –Τερματισμός [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 –Τερματισμός [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

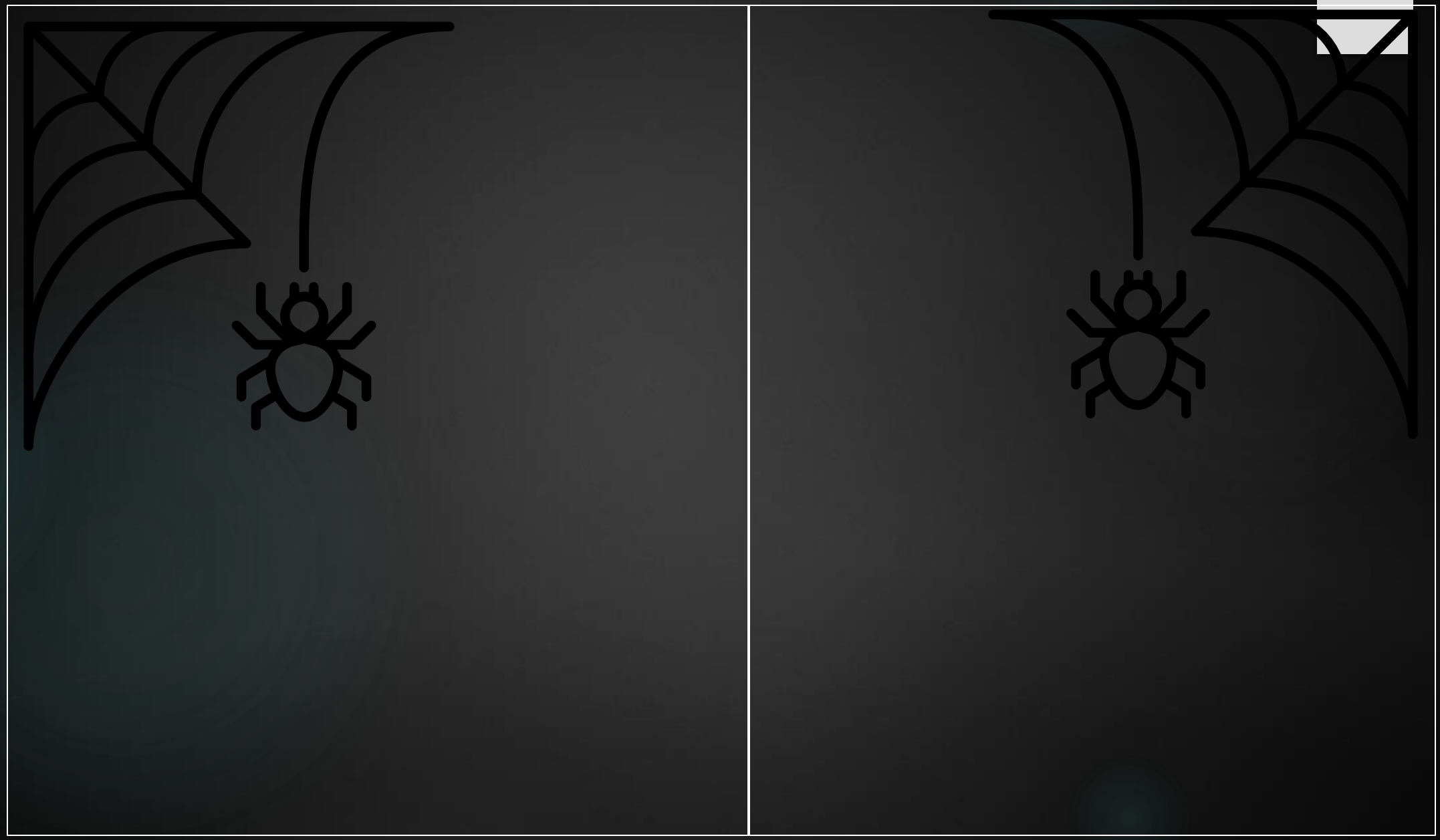
Άσκηση 2 –Τερματισμός [count = 1]

```
while(1) {
        pthread_mutex_lock(&mutex);
        while(count == 1) {
            pthread_cond_signal(&condition);
            pthread_cond_wait(&condition, &mutex);
        }

        scanf("%d", &number);
        count += 1;
        if (number == -1) {
            terminate = true;

            while(terminated != num_workers) {
                pthread_cond_signal(&condition);
                pthread_mutex_unlock(&mutex);
                pthread_mutex_lock(&mutex);
            }
            return 0;
        }
        pthread_mutex_unlock(&mutex);
    }
```

Άσκηση 2 –Τερματισμός [count = 1]



Άσκηση 3 –Ψευδοκώδικας empty και αλλαγή προτεραιοτητας

```
Mutex lock
.
.
 --car_counter;
.
.
if(car_counter == -1)
  if(blue_turn == true) {
      blue_turn = false;
      pthread_cond_signal(&s_red);
   }else if(red_turn == true){
      red_turn = false;
      pthread_cond_signal(&s_blue);
   }
  pthread_cond_wait(&empty, &mutex);
}
car_counter--;
if(car_counter >= max_cars){
   pthread_cond_signal(&full);
}

Mutex unlock
```

```
Mutex lock
.
.
while(red_turn == true) {
    pthread_cond_wait(&s_blue, &mutex);
}
    pthread_cond_signal(&s_blue);
    blue_turn = true;
.
.
++car_counter;
.
.
if(car_counter == 0){
     car_counter++;
     pthread_cond_signal(&empty);
    }
.
.
Mutex unlock
```

```
Mutex lock
.
.
while(blue_turn == true) {
    pthread_cond_wait(&s_red, &mutex);
}
    pthread_cond_signal(&s_red);
    red_turn = true;
.
.
++car_counter;
.
.
if(car_counter == 0){
     car_counter++;
     pthread_cond_signal(&empty);
    }
.
.
Mutex unlock
```

Άσκηση 3 –Ψευδοκώδικας blues turn red waiting

```
Mutex lock
.
.
 --car_counter;
.
.
if(car_counter == -1)
  if(blue_turn == true) {
      blue_turn = false;
      pthread_cond_signal(&s_red);
   }else if(red_turn == true){
      red_turn = false;
      pthread_cond_signal(&s_blue);
   }
  pthread_cond_wait(&empty, &mutex);
}
car_counter--;
if(car_counter >= max_cars){
   pthread_cond_signal(&full);
}

Mutex unlock
```

```
Mutex lock
.
.
while(red_turn == true) {
    pthread_cond_wait(&s_blue, &mutex);
}
    pthread_cond_signal(&s_blue);
    blue_turn = true;
.
.
++car_counter;
.
.
if(car_counter == 0){
     car_counter++;
     pthread_cond_signal(&empty);
    }
.
.
Mutex unlock
```

```
Mutex lock
.
.
while(blue_turn == true) {
    pthread_cond_wait(&s_red, &mutex);
}
    pthread_cond_signal(&s_red);
    red_turn = true;
.
.
++car_counter;
.
.
if(car_counter == 0){
     car_counter++;
     pthread_cond_signal(&empty);
    }
.
.
Mutex unlock
```

Άσκηση 3 –Ψευδοκώδικας ο consumer δίνει ελεύθερο και στα δυο

```
Mutex lock
.
.
 --car_counter;
.
.
if(car_counter == -1)
  if(blue_turn == true) {
      blue_turn = false;
      pthread_cond_signal(&s_red);
   }else if(red_turn == true){
      red_turn = false;
      pthread_cond_signal(&s_blue);
   }
  pthread_cond_wait(&empty, &mutex);
}
car_counter--;
if(car_counter >= max_cars){
   pthread_cond_signal(&full);
}

Mutex unlock
```

```
Mutex lock
.
.
while(red_turn == true) {
    pthread_cond_wait(&s_blue, &mutex);
}
    pthread_cond_signal(&s_blue);
    blue_turn = true;
.

++car_counter;
.
.
if(car_counter == 0){
     car_counter++;
     pthread_cond_signal(&empty);
    }
.
.
Mutex unlock
```

```
Mutex lock
.
.
while(blue_turn == true) {
    pthread_cond_wait(&s_red, &mutex);
}
    pthread_cond_signal(&s_red);
    red_turn = true;
.
.
++car_counter;
.
.
if(car_counter == 0){
     car_counter++;
     pthread_cond_signal(&empty);
    }
.
.
Mutex unlock
```

Άσκηση 3 –Ψευδοκώδικας δείχνει ότι η γέφυρα είναι άδεια

```
Mutex lock
.
.
 --car_counter;
.
.
if(car_counter == -1)
  if(blue_turn == true) {
      blue_turn = false;
      pthread_cond_signal(&s_red);
   }else if(red_turn == true){
      red_turn = false;
      pthread_cond_signal(&s_blue);
   }
  pthread_cond_wait(&empty, &mutex);
}
car_counter--;
if(car_counter >= max_cars){
   pthread_cond_signal(&full);
}

Mutex unlock
```

```
Mutex lock
.
.
while(red_turn == true) {
    pthread_cond_wait(&s_blue, &mutex);
}
    pthread_cond_signal(&s_blue);
    blue_turn = true;
.
.
++car_counter;
.
.
if(car_counter == 0){
     car_counter++;
     pthread_cond_signal(&empty);
    }
.
.
Mutex unlock
```

```
Mutex lock
.
.
while(blue_turn == true) {
    pthread_cond_wait(&s_red, &mutex);
}
    pthread_cond_signal(&s_red);
    red_turn = true;
.
.
++car_counter;
.
.
if(car_counter == 0){
     car_counter++;
     pthread_cond_signal(&empty);
    }
.
.
Mutex unlock
```

Άσκηση 3 –Ψευδοκώδικας παίρνει το mutex το blue

```
Mutex lock
.
.
 --car_counter;
.
.
if(car_counter == -1)
  if(blue_turn == true) {
      blue_turn = false;
      pthread_cond_signal(&s_red);
   }else if(red_turn == true){
      red_turn = false;
      pthread_cond_signal(&s_blue);
   }
  pthread_cond_wait(&empty, &mutex);
}
car_counter--;
if(car_counter >= max_cars){
   pthread_cond_signal(&full);
}

Mutex unlock
```

```
Mutex lock
.
.
while(red_turn == true) {
    pthread_cond_wait(&s_blue, &mutex);
}
    pthread_cond_signal(&s_blue);
    blue_turn = true;
.
.
++car_counter;
.
.
if(car_counter == 0){
     car_counter++;
     pthread_cond_signal(&empty);
    }
.
.
Mutex unlock
```

```
Mutex lock
.
.
while(blue_turn == true) {
    pthread_cond_wait(&s_red, &mutex);
}
    pthread_cond_signal(&s_red);
    red_turn = true;
.
.
++car_counter;
.
.
if(car_counter == 0){
     car_counter++;
     pthread_cond_signal(&empty);
    }
.
.
Mutex unlock
```

Άσκηση 3 –Ψευδοκώδικας δίνει προτεραιότητα στα εαυτό του

```
Mutex lock
.
.
 --car_counter;
.
.
if(car_counter == -1)
  if(blue_turn == true) {
      blue_turn = false;
      pthread_cond_signal(&s_red);
   }else if(red_turn == true){
      red_turn = false;
      pthread_cond_signal(&s_blue);
   }
  pthread_cond_wait(&empty, &mutex);
}
car_counter--;
if(car_counter >= max_cars){
   pthread_cond_signal(&full);
}

Mutex unlock
```

```
Mutex lock
.
.
while(red_turn == true) {
    pthread_cond_wait(&s_blue, &mutex);
}
    pthread_cond_signal(&s_blue);
    blue_turn = true;
.
.
++car_counter;
.
.
if(car_counter == 0){
     car_counter++;
     pthread_cond_signal(&empty);
    }
.
.
Mutex unlock
```

```
Mutex lock
.
.
while(blue_turn == true) {
    pthread_cond_wait(&s_red, &mutex);
}
    pthread_cond_signal(&s_red);
    red_turn = true;
.
.
++car_counter;
.
.
if(car_counter == 0){
     car_counter++;
     pthread_cond_signal(&empty);
    }
.
.
Mutex unlock
```

Άσκηση 3 –Ψευδοκώδικας Δίνει σήμα ότι δεν είναι άδεια η γέφυρα

```
Mutex lock
.
.
 --car_counter;
.
.
if(car_counter == -1)
  if(blue_turn == true) {
      blue_turn = false;
      pthread_cond_signal(&s_red);
   }else if(red_turn == true){
      red_turn = false;
      pthread_cond_signal(&s_blue);
   }
  pthread_cond_wait(&empty, &mutex);
}
car_counter--;
if(car_counter >= max_cars){
   pthread_cond_signal(&full);
}

Mutex unlock
```

```
Mutex lock
.
.
while(red_turn == true) {
    pthread_cond_wait(&s_blue, &mutex);
}
    pthread_cond_signal(&s_blue);
    blue_turn = true;
.
.
++car_counter;
.
.
if(car_counter == 0){
     car_counter++;
     pthread_cond_signal(&empty);
    }
.
.
Mutex unlock
```

```
Mutex lock
.
.
while(blue_turn == true) {
    pthread_cond_wait(&s_red, &mutex);
}
    pthread_cond_signal(&s_red);
    red_turn = true;
.
.
++car_counter;
.
.
if(car_counter == 0){
     car_counter++;
     pthread_cond_signal(&empty);
    }
.
.
Mutex unlock
```

Άσκηση 3 –Ψευδοκώδικας Παίρνει το mutex το red

```
Mutex lock
.
.
 --car_counter;
.
.
if(car_counter == -1)
  if(blue_turn == true) {
      blue_turn = false;
      pthread_cond_signal(&s_red);
   }else if(red_turn == true){
      red_turn = false;
      pthread_cond_signal(&s_blue);
   }
  pthread_cond_wait(&empty, &mutex);
}
car_counter--;
if(car_counter >= max_cars){
   pthread_cond_signal(&full);
}

Mutex unlock
```

```
Mutex lock
.
.
while(red_turn == true) {
    pthread_cond_wait(&s_blue, &mutex);
}
    pthread_cond_signal(&s_blue);
    blue_turn = true;
.
.
++car_counter;
.
.
if(car_counter == 0){
     car_counter++;
     pthread_cond_signal(&empty);
    }
.
.
Mutex unlock
```

```
Mutex lock
.
.
while(blue_turn == true) {
    pthread_cond_wait(&s_red, &mutex);
}
    pthread_cond_signal(&s_red);
    red_turn = true;
.
.
++car_counter;
.
.
if(car_counter == 0){
     car_counter++;
     pthread_cond_signal(&empty);
    }
.
.
Mutex unlock
```

Άσκηση 3 –Ψευδοκώδικας Με το while ελένχει αν έχει προτεραιότητα

```
Mutex lock
.
.
 --car_counter;
.
.
if(car_counter == -1)
  if(blue_turn == true) {
      blue_turn = false;
      pthread_cond_signal(&s_red);
   }else if(red_turn == true){
      red_turn = false;
      pthread_cond_signal(&s_blue);
   }
  pthread_cond_wait(&empty, &mutex);
}
car_counter--;
if(car_counter >= max_cars){
   pthread_cond_signal(&full);
}

Mutex unlock
```

```
Mutex lock
.
.
while(red_turn == true) {
    pthread_cond_wait(&s_blue, &mutex);
}
    pthread_cond_signal(&s_blue);
    blue_turn = true;
.
.
++car_counter;
.
.
if(car_counter == 0){
     car_counter++;
     pthread_cond_signal(&empty);
    }
.
.
Mutex unlock
```

```
Mutex lock
.
.
while(blue_turn == true) {
    pthread_cond_wait(&s_red, &mutex);
}
    pthread_cond_signal(&s_red);
    red_turn = true;
.
.
++car_counter;
.
.
if(car_counter == 0){
     car_counter++;
     pthread_cond_signal(&empty);
    }
.
.
Mutex unlock
```

Άσκηση 3 –Ψευδοκώδικας Δεν έχει

```
Mutex lock
.
.
 --car_counter;
.
.
if(car_counter == -1)
  if(blue_turn == true) {
      blue_turn = false;
      pthread_cond_signal(&s_red);
   }else if(red_turn == true){
      red_turn = false;
      pthread_cond_signal(&s_blue);
   }
  pthread_cond_wait(&empty, &mutex);
}
car_counter--;
if(car_counter >= max_cars){
   pthread_cond_signal(&full);
}

Mutex unlock
```

```
Mutex lock
.
.
while(red_turn == true) {
    pthread_cond_wait(&s_blue, &mutex);
}
    pthread_cond_signal(&s_blue);
    blue_turn = true;
.
.
++car_counter;
.
.
if(car_counter == 0){
     car_counter++;
     pthread_cond_signal(&empty);
    }
.
.
Mutex unlock
```

```
Mutex lock
.
.
while(blue_turn == true) {
    pthread_cond_wait(&s_red, &mutex);
}
    pthread_cond_signal(&s_red);
    red_turn = true;
.
.
++car_counter;
.
.
if(car_counter == 0){
     car_counter++;
     pthread_cond_signal(&empty);
    }
.
.
Mutex unlock
```

Άσκηση 3 -Ψευδοκώδικας Συγχρονισμός full

```
Mutex lock
.
.
--car_counter;
if(car_counter > max_cars - 1) {
   pthread_cond_signal(&full);
}
.
.
Mutex unlock
```

```
Mutex lock
.
.
++car_counter;
if(car_counter > max_cars){
        pthread_cond_wait(&full, &mutex);
}
.
.
Mutex unlock
```

Άσκηση 3 –Ψευδοκώδικας Car βλέπει ότι η γέφυρα είναι full

```
Mutex lock
.
.
--car_counter;
if(car_counter > max_cars - 1) {
   pthread_cond_signal(&full);
}
.
.
Mutex unlock
```

```
Mutex lock
.
.
++car_counter;
if(car_counter > max_cars){
        pthread_cond_wait(&full, &mutex);
}
.
.
Mutex unlock
```

Άσκηση 3 -Ψευδοκώδικας Consumer ξεμπλοκάρει

```
Mutex lock
.
.
--car_counter;
if(car_counter > max_cars - 1) {
   pthread_cond_signal(&full);
}
.
.
Mutex unlock
```

```
Mutex lock
.
.
++car_counter;
if(car_counter > max_cars){
   pthread_cond_wait(&full, &mutex);
}
.
.
Mutex unlock
```

Άσκηση 3 -Ψευδοκώδικας Έχει το mutex το car 1

```
Mutex lock
.
.
--car_counter;
if(car_counter > max_cars - 1) {
   pthread_cond_signal(&full);
}
.
.
Mutex unlock
```

```
Mutex lock
.
.
++car_counter;
 full_again = true;
 while(car_counter > max_cars && full_again){
        pthread_cond_wait(&full, &mutex);
        if(car_on_brige == max_cars) {
            full_again = true;
        }else{
            full_again = false;
        }
 }
.
.
Mutex unlock
```

```
Mutex lock
.
.
++car_counter;
 full_again = true;
 while(car_counter > max_cars && full_again){
        pthread_cond_wait(&full, &mutex);
        if(car_on_brige == max_cars) {
            full_again = true;
        }else{
            full_again = false;
        }
 }
.
.
Mutex unlock
```

Άσκηση 3 -Ψευδοκώδικας ο consumer ξεμπλοκάρει το car 1

```
Mutex lock
.
.
--car_counter;
if(car_counter > max_cars - 1) {
    pthread_cond_signal(&full);
}
.
.
Mutex unlock
```

```
Mutex lock
.
.
++car_counter;
 full_again = true;
 while(car_counter > max_cars && full_again){
        pthread_cond_wait(&full, &mutex);
        if(car_on_brige == max_cars) {
            full_again = true;
        }else{
            full_again = false;
        }
 }
.
.
Mutex unlock
```

```
Mutex lock
.
.
++car_counter;
 full_again = true;
 while(car_counter > max_cars && full_again){
        pthread_cond_wait(&full, &mutex);
        if(car_on_brige == max_cars) {
            full_again = true;
        }else{
            full_again = false;
        }
 }
.
.
Mutex unlock
```

Άσκηση 3 -Ψευδοκώδικας Παίρνει το mutex το car 2

```
Mutex lock
.
.
--car_counter;
if(car_counter > max_cars - 1) {
   pthread_cond_signal(&full);
}
.
.
Mutex unlock
```

```
Mutex lock
.
.
++car_counter;
 full_again = true;
 while(car_counter > max_cars && full_a
        pthread_cond_wait(&full, &mutex);
        if(car_on_brige == max_cars) {
            full_again = true;
        }else{
            full_again = false;
        }
 }
.
.
Mutex unlock
```

```
Mutex lock
.
.
++car_counter;
 full_again = true;
 while(car_counter > max_cars && full_again){
        pthread_cond_wait(&full, &mutex);
        if(car_on_brige == max_cars) {
            full_again = true;
        }else{
            full_again = false;
        }
 }
.
.
Mutex unlock
```

Άσκηση 3 -Ψευδοκώδικας Παίρνει το mutex το car 1

```
Mutex lock
.
.
--car_counter;
if(car_counter > max_cars - 1) {
   pthread_cond_signal(&full);
}
.
.
Mutex unlock
```

```
Mutex lock
.
.
++car_counter;
 full_again = true;
 while(car_counter > max_cars && full_again){
        pthread_cond_wait(&full, &mutex);
        if(car_on_brige == max_cars) {
            full_again = true;
        }else{
            full_again = false;
        }
 }
.
.
Mutex unlock
```

```
Mutex lock
.
.
++car_counter;
 full_again = true;
 while(car_counter > max_cars && full_again){
        pthread_cond_wait(&full, &mutex);
        if(car_on_brige == max_cars) {
            full_again = true;
        }else{
            full_again = false;
        }
 }
.
.
Mutex unlock
```

Άσκηση 3 -Ψευδοκώδικας Περίπτωση που ξαναμπλοκάρει full

```
Mutex lock
.
.
--car_counter;
if(car_counter > max_cars - 1) {
   pthread_cond_signal(&full);
}
.
.
Mutex unlock
```

```
Mutex lock
.
.
++car_counter;
 full_again = true;
 while(car_counter > max_cars && full_again){
        pthread_cond_wait(&full, &mutex);
        if(car_on_brige == max_cars) {
            full_again = true;
        }else{
            full_again = false;
        }
 }
.
.
Mutex unlock
```

```
Mutex lock
.
.
++car_counter;
 full_again = true;
 while(car_counter > max_cars && full_again){
        pthread_cond_wait(&full, &mutex);
        if(car_on_brige == max_cars) {
            full_again = true;
        }else{
            full_again = false;
        }
 }
.
.
Mutex unlock
```

Άσκηση 3 -Ψευδοκώδικας Περίπτωση που ξαναμπλοκάρει full

```
Mutex lock
.
.
--car_counter;
if(car_counter > max_cars - 1) {
   pthread_cond_signal(&full);
}
.
.
Mutex unlock
```

```
Mutex lock
.
.
++car_counter;
 full_again = true;
 while(car_counter > max_cars && full_again){
        pthread_cond_wait(&full, &mutex);
        if(car_on_brige == max_cars) {
            full_again = true;
        }else{
            full_again = false;
        }
 }
.
.
Mutex unlock
```

```
Mutex lock
.
.
++car_counter;
 full_again = true;
 while(car_counter > max_cars && full_again){
        pthread_cond_wait(&full, &mutex);
        if(car_on_brige == max_cars) {
            full_again = true;
        }else{
            full_again = false;
        }
 }
.
.
Mutex unlock
```

Άσκηση 3 -Ψευδοκώδικας Περίπτωση που ξαναμπλοκάρει full

```
Mutex lock
.
.
--car_counter;
if(car_counter > max_cars - 1) {
   pthread_cond_signal(&full);
}
.
.
Mutex unlock
```

```
Mutex lock
.
.
++car_counter;
 full_again = true;
 while(car_counter > max_cars && full_again){
        pthread_cond_wait(&full, &mutex);
        if(car_on_brige == max_cars) {
            full_again = true;
        }else{
            full_again = false;
        }
 }
.
.
Mutex unlock
```

```
Mutex lock
.
.
++car_counter;
 full_again = true;
 while(car_counter > max_cars && full_again){
        pthread_cond_wait(&full, &mutex);
        if(car_on_brige == max_cars) {
            full_again = true;
        }else{
            full_again = false;
        }
 }
.
.
Mutex unlock
```

Άσκηση 3 -Ψευδοκώδικας Περίπτωση που ξαναμπλοκάρει full

```
Mutex lock
.
.
--car_counter;
if(car_counter > max_cars - 1) {
   pthread_cond_signal(&full);
}
.
.
Mutex unlock
```

```
Mutex lock
.
.
++car_counter;
 full_again = true;
 while(car_counter > max_cars && full_again){
        pthread_cond_wait(&full, &mutex);
        if(car_on_brige == max_cars) {
            full_again = true;
        }else{
            full_again = false;
        }
 }
.
.
Mutex unlock
```

```
Mutex lock
.
.
++car_counter;
 full_again = true;
 while(car_counter > max_cars && full_again){
        pthread_cond_wait(&full, &mutex);
        if(car_on_brige == max_cars) {
            full_again = true;
        }else{
            full_again = false;
        }
 }
.
.
Mutex unlock
```

Άσκηση 3 –Ψευδοκώδικας Περίπτωση που ξαναμπλοκάρει full

```
Mutex lock
.
.
--car_counter;
if(car_counter > max_cars - 1) {
   pthread_cond_signal(&full);
}
.
.
Mutex unlock
```

```
Mutex lock
.
.
++car_counter;
 full_again = true;
 while(car_counter > max_cars && full_again){
        pthread_cond_wait(&full, &mutex);
        if(car_on_brige == max_cars) {
            full_again = true;
        }else{
            full_again = false;
        }
 }
.
.
Mutex unlock
```

```
Mutex lock
.
.
++car_counter;
 full_again = true;
 while(car_counter > max_cars && full_again){
        pthread_cond_wait(&full, &mutex);
        if(car_on_brige == max_cars) {
            full_again = true;
        }else{
            full_again = false;
        }
 }
.
.
Mutex unlock
```

Άσκηση 4 -Ψευδοκώδικας

```
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=0]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

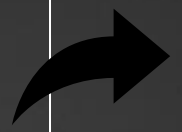
Άσκηση 4 –Κανονική λειτουργία [boarded=0]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=0]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```



```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

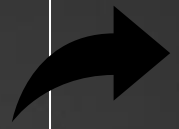
Άσκηση 4 –Κανονική λειτουργία [boarded=0]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=0]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```



```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=1]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=1]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);    }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);     }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=max-1]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=max-1]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=max-1]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=max-1]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=max]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=max]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=max]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=max]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=max]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);    }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=max]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

(δεν έχει σημασία η σειρά αν ξυπνήσει πρώτος ο τελευταίος ή αυτοί που περιμένουν)

Άσκηση 4 –Κανονική λειτουργία [boarded=max]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=max]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=max]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlock
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=max]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=max]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Περίπτωση που παίρνει ξανά passenger το mutex ή το πήρε από το signal για το τρένο

Άσκηση 4 –Κανονική λειτουργία [boarded=max]

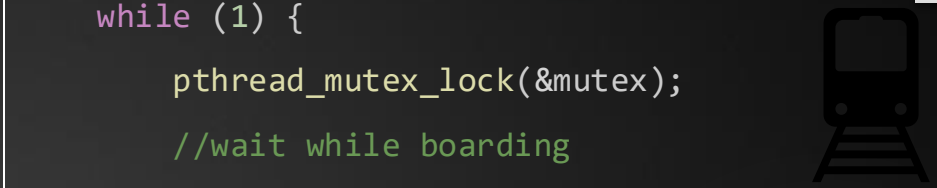
```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=max]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);    }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```



Άσκηση 4 –Κανονική λειτουργία [boarded=max]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);    }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=max]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=max]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=max]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Το mutex πήγε σε καινούργιο επιβάτη! Δεν υπάρχει θέμα.

Άσκηση 4 –Κανονική λειτουργία [boarded=max]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=max]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

*(το mutex πηγαίνει στο τρένο μετά από μερικές επαναλήψεις)

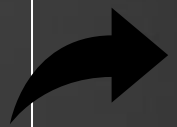
Άσκηση 4 –Κανονική λειτουργία [boarded=max]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=max]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```



```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=max]

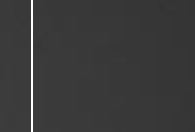
```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

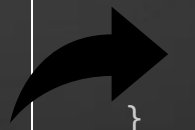
*Mutex πηγαίνει ξανά σε επιβάτη

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=max]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);    }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```





```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=max]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

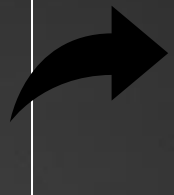
Άσκηση 4 –Κανονική λειτουργία [boarded=max]

```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=0]

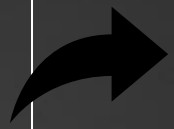
```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```



```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=0]

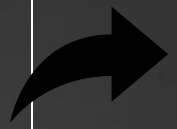
```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);    }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```



```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=0]

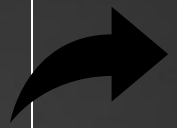
```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);    }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```



```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=0]

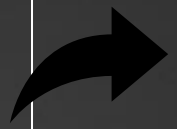
```c
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```



```c
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Άσκηση 4 –Κανονική λειτουργία [boarded=0]

```
void *passenger_thread(void *args) {
    pthread_mutex_lock(&mutex);
     //if signaled but passengers = full wait
    while(boarded == *max_passenger){
        // unlock mutex until train unlocks
        pthread_mutex_unlock(&mutex);
        pthread_mutex_lock(&mutex);   }
    if(boarded < *max_passenger -1) {
        boarded++;
        // wait for the ride to finish
        pthread_cond_wait(&train_finish,&mutex);    }
    // If you are the last passenger notify the train
    else if(boarded == *max_passenger -1) {
        boarded++;
        pthread_cond_signal(&train_wait);    }
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```



```
void *train_thread(void *args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        //wait while boarding
        boarded = 0;
        pthread_cond_wait(&train_wait, &mutex);
        if (end == true) { break; }
        //ride
        sleep(*);
        //passengers leaving the ride
        for (i = boarded; i > 0; i--) {
            pthread_cond_signal(&train_finish);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Φτάσαμε στην αρχική κατάσταση!