На наших тестовых данных решение сходу не заработало. На устройстве приложение вылетает из-за нехватки памяти, на эмуляторе висит полуживое из-за 100% загрузки главного потока. Иногда падает с нарушением доступа к памяти. Причины ниже.

Организация парсинга следующая: получает очередной кусок данных из сети и порождает posix thread с набором данных для парсинга

```
bool CLogReader::AddSourceBlock(const char *block, const size_t block_size) {
    if (block == NULL || block_size == 0)
    {
        printf("Warning! invalid arguments");
        return false;
    }

    if (vars::search_key == NULL) {
        printf("Warning! the search_key has null value.");
        return false;
    }
    // making copy of block into BlockInfo
    char *mem = (char *)malloc(block_size + 1);
    if (mem == NULL)
        return false;
    memcpy(mem, block, block_size);

    char *key = NULL;
    size_t key_size = strlen(vars::search_key);
    key = (char *)malloc(key_size + 1);
    if (key == NULL) {
        free(mem);
        return false;
    }
    memcpy(key, vars::search_key, key_size);

    BlockInfo *info = (BlockInfo *)malloc(sizeof(BlockInfo));
    info->data = mem;
    info->key = key;
    info->size = block_size;

    // prepare ThreadInfo structure
    bool result = api::LaunchThread(info);
```

```
    if (!result) {
        info->free();
    }
    return result;
}
```

Внутри функции потока:

```
    void *FindMatchesInLines(void *arg) {
        BlockInfo * info = (BlockInfo *)arg;
        pthread_mutex_lock(&vars::mutex);
        printf("Started thread number %d\n", info->thread_counter);

        // getting appendix_size if it stored from previous block
        // the urlsession loads log file dividing in to parts that ends not '\n'.
        // In this case previous part contains not completed line and next part begins from final part of line.
        char *buffer = NULL;
        if (vars::appendix)
        {
            size_t appendix_size = strlen(vars::appendix);
            buffer = (char *)malloc(appendix_size + info->size + 1); //allocated memory
            memcpy(buffer, vars::appendix, appendix_size);
            memcpy(&buffer[appendix_size], info->data, info->size);
            free(vars::appendix);
            vars::appendix = NULL;
        }

        if (buffer == NULL)
        {
            buffer = (char *)malloc(info->size);
            memcpy(buffer, info->data, info->size);
        }
        // search for lines in the buffer
        char *freed_buffer = buffer;
        char *line_ptr = NULL;
        char *last_line_ptr = NULL;
```

```
        while( (line_ptr = strsep(&buffer, "\n")) != NULL )
        {
            if (IsLineMatchToKey(line_ptr, info->key)) {
                api::call_back(line_ptr);
            }
            last_line_ptr = line_ptr;
        }

        // store appendix if last line is not completed.
        size_t size_last_line = strlen(last_line_ptr);
        if (last_line_ptr[size_last_line] != '\n' && vars::appendix == NULL)
        {
            vars::appendix = strdup(last_line_ptr);
}
```

Строка для strsep нигде не терминируется (и паттерн для парсинга тоже), а значит может быть выход за границу блока, что периодически происходит на эмуляторе.

Последняя неполная строка запоминается в общем хранилище для всех потоков-обработчиков. Поскольку порядок запуска потоков не гарантируется, возможна ситуация (маловероятная), когда остаток от блока n-1 заберет n+1 поток, а не n-ный.

Метод обработки результатов

```
- (void)reader:(nullable LogReader *)reader foundLines:(nullable NSString *)lines {
    if (lines == nil) return;
    __weak typeof (self) wself = self;
    dispatch_async(_queue, ^{
        [wself.model appendString:lines];
        [wself.model appendString:@"\n"];
        [wself.model appendString:@"\n"];
        [wself saveToLogFile:lines];
        [wself updateLogView];
    });
}
- (void)updateLogView {
    __weak typeof (self) wself = self;
    if (0 < _counter--) return;
    _counter = MAX_UPDATE_COUNTER;
```

```
    dispatch_async(dispatch_get_main_queue(), ^{
        wself.textView.text = wself.model;
        if (wself.inProgress == YES)
            wself.inProgress = NO;
        if (wself.isResultReady == NO)
            wself.isResultReady = YES;
    });
}
```

Во-первых, записывает и читает LogWindowViewController.model из разных очередей.

Во-вторых, на каждую найденную строку ставит отдельную задачу + целиком обновляет полный текст результатов в TextView. Поэтому загрузка 100% главного потока.

Сам алгоритм парсинга взял готовый, о чем честно указал

```
    bool IsLineMatchToKey(const char * line, const char * key) {
        // Written by Jack Handy - <A
href="mailto:jakkhandy@hotmail.com">jakkhandy@hotmail.com</A>
        const char *cp = NULL, *mp = NULL;

        while ((*line) && (*key != '*')) {
            if ((*key != *line) && (*key != '?')) {
                return 0;
            }
            key++;
            line++;
        }

        while (*line) {
            if (*key == '*') {
                if (!*++key) {
                    return 1;
                }
                mp = key;
                cp = line+1;
            } else if ((*key == *line) || (*key == '?')) {
                key++;
```

```
            line++;
        } else {
            key = mp;
            line = cp++;
        }
    }

    return !*key;
}
```

Код читается тяжело, сильный оверинжиниринг.

Учитывая все перечисленное, а также то, что у кандидата было время все проверить (начальный коммит на гитхабе от 16 июля) - отказ.